CURSO PRÁTICO 2 DE PROGRAMAÇÃO DE COMPUTADORES

INPUT

PROGRAMAÇÃO BASIC - PROGRAMAÇÃO DE JOGOS - CÓDIGO DE MÁQUINA

NOVA CULTURAL

Cz$ 20,00

INPUT

Vol. 1 Nº 2

NESTE NÚMERO

PROGRAMAÇÃO BASIC

A ARTE DE FAZER LAÇOS

A utilidade dos comandos FOR..NEXT. Como criar efeitos sonoros. Faça o sol se pôr, movendo apenas algumas teclas e desvende a beleza dos caleidoscópios. Como provocar repetições de um trecho de programa 21

PROGRAMAÇÃO DE JOGOS

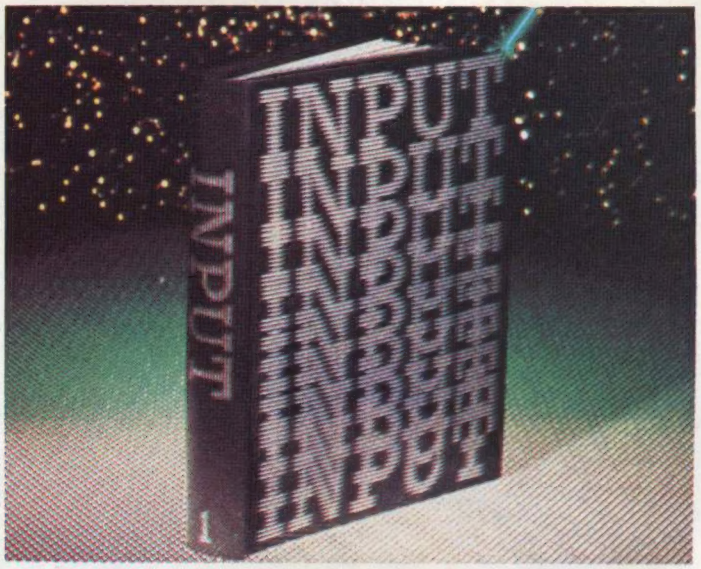
APONTAR... FOGO!

Aprenda a verificar se uma tecla foi pressionada e a controlar figuras em movimento na tela. Para que serve o comando INKEY$? Dispare seu míssil e destrua o inimigo. Veja para que serve a auto-repetição 29

CÓDIGO DE MÁQUINA

APRENDA A CONTAR COM UM DEDO SÓ

Alguns sistemas numéricos. Faça contas em diferentes bases. O sistema binário constitui o idioma do computador. Uma calculadora para todas as bases. Bits e Bytes. Para poder programar em linguagem de máquina, é necessário conhecer o sistema binário 34



PLANO DA OBRA

"INPUT" é uma obra editada em fascículos semanais, e cada conjunto de 15 fascículos compõe um volume. A capa para encadernação de cada volume estará à venda oportunamente.

COMPLETE SUA COLEÇÃO

Exemplares atrasados, até seis meses após o encerramento da coleção, poderão ser comprados, a preços atualizados, da seguinte forma: **1. Pessoalmente** — por meio de seu jornaleiro ou dirigindo-se ao distribuidor local, cujo endereço poderá ser facilmente conseguido junto a qualquer jornaleiro de sua cidade. Em São Paulo os endereços são: Rua Brigadeiro Tobias, 773 (Centro); Av. Industrial, 117 (Santo André); e, no Rio de Janeiro: Rua da Passagem, 93 (Botafogo). **2. Por carta** — Poderão ser solicitados exemplares atrasados também por carta, que deve ser enviada para DINAP — Distribuidor Nacional de Publicações — Números Atrasados — Estrada Velha de Osasco, 132 (Jardim Tereza) — CEP 06000 — Osasco — São Paulo. **3. Por telex** — Utilize o nº (011) 33670 ABSA. Em **Portugal**, os pedidos devem ser feitos à Distribuidora Jardim de Publicações Ltd. — Qta. Pau Varais, Azinhaga de Fetais — 2685, Camarate — Lisboa; Tel. 257-2542 — Apartado 57 — Telex 43 069 JARLIS P.
Não envie pagamento antecipado. O atendimento será feito pelo reembolso postal e o pagamento, incluindo as despesas postais, deverá ser efetuado ao se retirar a encomenda na Agência do Correio. **Atenção**: Após seis meses do encerramento da coleção, os pedidos serão atendidos, dependendo da disponibilidade de estoque. **Obs.:** Quando pedir livros, mencione sempre o título e/ou o autor da obra, além do número da edição.

COLABORE CONOSCO

Encaminhe seus comentários, críticas, sugestões ou reclamações ao Serviço de Atendimento ao Leitor — Caixa Postal 9442, São Paulo — SP.

Editor
VICTOR CIVITA

REDAÇÃO

Diretora Editorial: Iara Rodrigues

Editor chefe: Paulo de Almeida
Editor de texto: Cláudio A.V. Cavalcanti
Editor de Arte: Eduardo Barreto
Chefe de Arte: Carlos Luiz Batista
Assistentes de Arte: Ailton Oliveira Lopes, Dilvacy M. Santos, José Maria de Oliveira, Grace A. Arruda, Monica Lenardon Corradi
Secretária de Redação/Coordenadora: Stefania Crema
Secretários de Redação: Beatriz Hagström, José Benedito de Oliveira Damião, Maria de Lourdes Carvalho, Marisa Soares de Andrade, Mauro de Queiroz
Secretário Gráfico: Antonio José Filho

COLABORADORES
Consultor Editorial Responsável: Dr. Renato M.E. Sabbatini (Diretor do Núcleo de Informática Biomédica da Universidade Estadual de Campinas)
Execução Editorial: DATAQUEST Assessoria em Informática Ltda. Campinas, SP.
Tradução: Aluísio J. Dornellas de Barros, Maria Fernanda Sabbatini
Adaptação, programação e redação: Aluísio J. Dornellas de Barros, Marcelo R. Pires Therezo, Raul Neder Porrelli
Coordenação geral: Rejane Felizatti Sabbatini
Assistente de Arte: Dagmar Bastos Sampaio

COMERCIAL
Diretor Comercial: Roberto Martins Silveira
Gerente Comercial: Flávio Ferrucio Maculan
Gerente de Circulação: Denise Maria Mozol

PRODUÇÃO
Gerente de Produção: João Stungis
Coordenador de Impressão: Atilio Roberto Bonon
Preparador de Texto/Coordenador: Eliel Silveira Cunha

Preparadores de Texto: Ana Maria Dilguerian, Antonio Francelino de Oliveira, Karina Ap. V. Grechi, Levon Yacubian, Maria Teresa Galluzzi, Paulo Felipe Mendrone
Revisor/Coordenador: José Maria de Assis
Revisoras: Conceição Aparecida Gabriel, Isabel Leite de Camargo, Ligia Aparecida Ricetto, Maria do Carmo Leme Monteiro, Maria Luiza Simões, Maria Teresa Martins Lopes.

© Marshall Cavendish Limited, 1984/85.
© Editora Nova Cultural Ltda., São Paulo, Brasil, 1986.

Edição organizada pela Editora Nova Cultural Ltda.
(Artigo 15 da Lei 5 988, de 14/12/1973).
Esta obra foi composta na AM Produções Gráficas Ltda. e impressa na Divisão Gráfica da Editora Abril S.A.

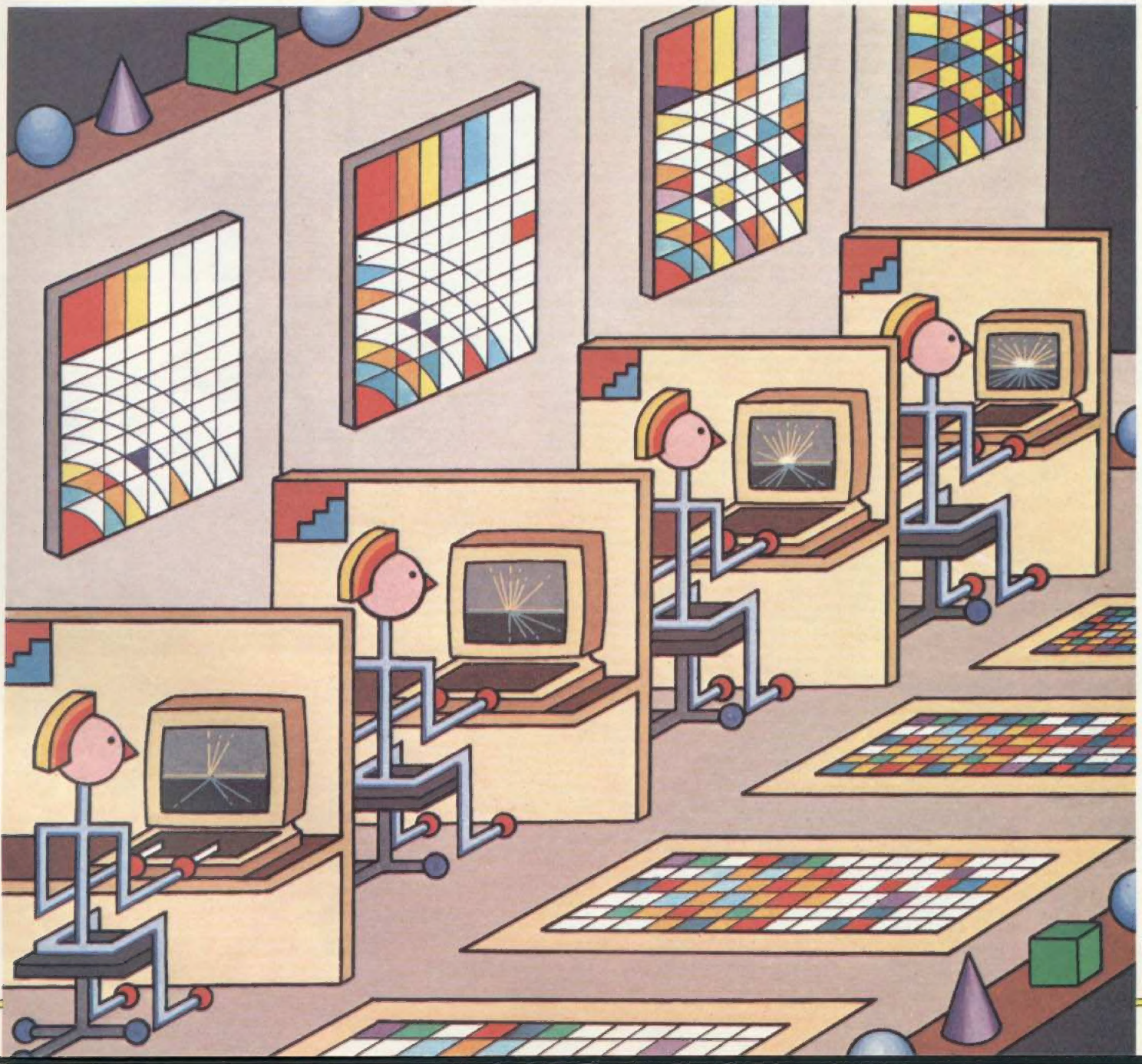
A ARTE DE FAZER LAÇOS

- APRENDA A CRIAR EFEITOS SONOROS
- DESENHE NÚMEROS COM LAÇOS FOR... NEXT
- O NASCER E O PÔR-DO-SOL NO HORIZONTE DO VÍDEO
- CALEIDOSCÓPIOS

Os comandos FOR e NEXT servem para provocar repetições de um trecho de programa em BASIC. Muito versáteis, eles são "pau para toda obra" e podem ser empregados em quase tudo, desde jogos até programas comerciais.

Um laço é uma estrutura de programação usada em operações repetitivas. Ele é usado quando se quer que o computador conte até um certo número, executando, ao mesmo tempo, alguma outra operação. Além disso, o laço pode ser aplicado na repetição do mesmo padrão gráfico em diferentes posições na tela. A combinação de instruções **FOR** e **NEXT** na linguagem BASIC é usada na programação de diversos tipos de laços.

Assim, você aprenderá a criar "pinturas instantâneas" usando laços **FOR... NEXT** em programas muito pequenos. Esses laços serão igualmente úteis na programação de jogos e em programas de todos os tipos.



O QUE É UM LAÇO FOR... NEXT

Um laço **FOR... NEXT** em BASIC é um mecanismo que faz com que o computador repita uma mesma operação certo número de vezes.

Suponha, por exemplo, que você quisesse saber as raízes quadradas de todos os números de 1 a 100. Você poderia dizer ao computador:

```
PRINT SQR(1)
PRINT SQR(2)
PRINT SQR(3)
```

... e assim por diante.

E a cada pergunta o computador daria a resposta. Mas, além de impor um enorme trabalho de digitação aos seus dedos, esta não é uma técnica particularmente eficiente de uso do computador. Tente assim:

```
10 FOR N=1 TO 100
20 PRINT N,SQR(N)
30 NEXT N
40 PRINT "E ACABOU."
```

Esse programa faz com que o computador imprima 1 e sua raiz quadrada, 2 e sua raiz quadrada, 3 e sua raiz quadrada.., e assim sucessivamente até atingir 100, no momento em que o aparelho pára.

Como ele faz isto? Quando o computador encontra um **FOR**, "sabe" que as linhas seguintes serão repetidas. Assim, ele as executa até encontrar o **NEXT**, volta à linha da instrução **FOR** e repete o processo.

Enquanto trabalha, o computador também está contando. A primeira vez em que passa pela linha 20 ele calcula a raiz quadrada de 1, na segunda, a de 2, e assim por diante.

Quando tiver trabalhado com o maior número da instrução **FOR**, ele deixará automaticamente o laço e continuará a execução do programa pela instrução seguinte ao **NEXT** — no nosso caso o **PRINT** da linha 40.

FRAÇÕES TAMBÉM

Ao executar um laço **FOR... NEXT**, o computador pode contar em unidades diferentes de 1. Para isso, usamos a declaração **STEP** (isto é "passo" ou "degrau") junto com a declaração **FOR**. Veja o exemplo:

```
10 FOR N=1 TO 30 STEP 2.7
20 PRINT N,SQR(N)
30 NEXT N
```

O computador não se atrapalha com o fato de que 30 não se divide em um número exato de passos, como você pediu. Ele vai o mais perto que pode e então pára.

O número de linhas entre o **FOR** e o **NEXT** tampouco o preocupa. Você pode colocar o **FOR** na linha 10 e o **NEXT** na 90 — ou mesmo 9000 — que ele não se esquecerá delas. Lembre-se, porém, de que o computador executará as linhas do laço a cada passagem.

COMO RETARDAR A AÇÃO

O laço **FOR... NEXT** tem dezenas de usos em programação. A mais simples delas é gastar tempo.

Se você rever a seção *Você sabe tabuada?* na primeira lição de BASIC, encontrará um bom exemplo. Nesse caso, tudo o que acontece entre cada **FOR** e o seu **NEXT** é que o computador "conta" um número. Essa contagem é muito rápida (o tempo exato pode ser da ordem de centésimos ou milésimos de segundo). Mas, enquanto espera que ele conte até 1000, você terá uma pausa bem perceptível. E se você executar:

```
10 FOR N=1 TO 1000000
20 NEXT N
```

... terá tempo, provavelmente, para tomar um café antes que ele termine.

APELO SONORO

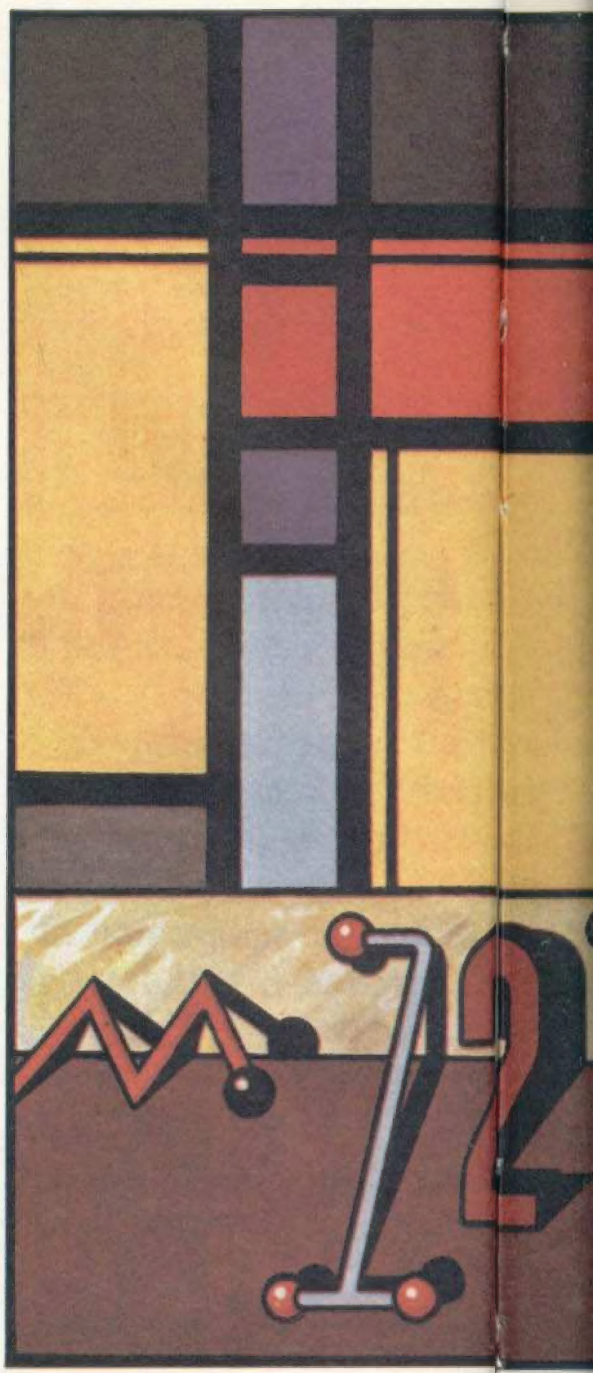
Programadores de jogos freqüentemente tornam essas pausas menos cansativas, inserindo nelas algumas notas musicais. Tente este exemplo:

```
10 FOR I=255 TO 155 STEP -1
20 SOUND I,1
30 NEXT I
```

```
10 FOR n=29 TO 10 STEP -1
20 SOUND .015,n
30 NEXT n
```

```
5 SOUND 7,56:SOUND 8,15
10 FOR I=255 TO 0 STEP -1
20 SOUND 0,I
30 NEXT I
```

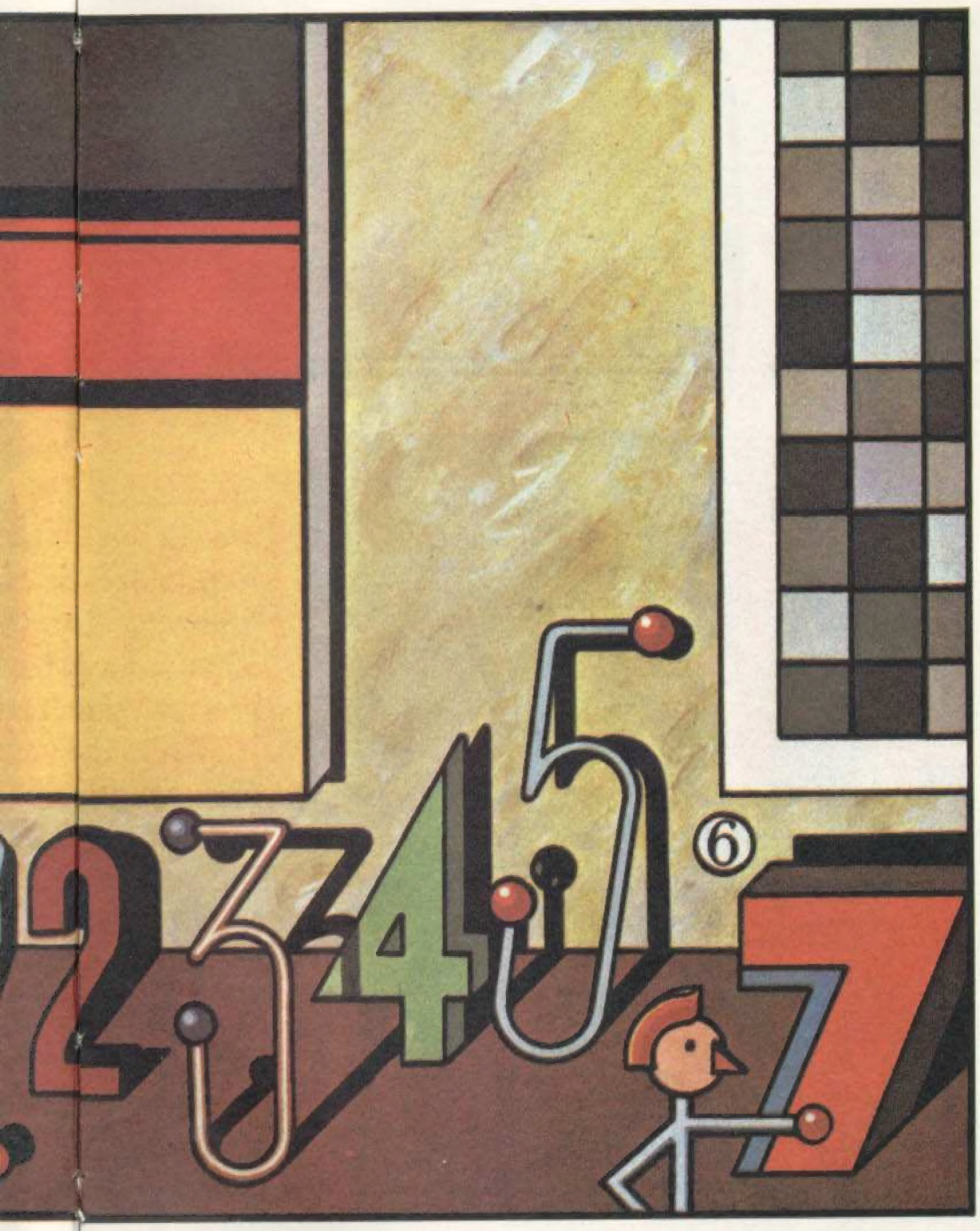
```
10  FOR N = 1 TO 100
20  PRINT  PEEK ( - 16336).
30  NEXT N
```



O Apple II padrão não têm comandos intrínsecos para a produção de efeitos sonoros. Desse modo, o efeito conseguido é apenas uma série de cliques. Já o TK-2000 tem o comando **SOUND**. Para rodar o programa acima no TK-2000, substitua as linhas 10 e 20 assim:

```
10 FOR I=80 TO 15 STEP -2
20 SOUND I,30
```

Este é um exemplo do efeito de som que os programadores usam em jogos para dizer "Você falhou" ou "O alie-



nígena aterrisou''. Ao mesmo tempo, quando se faz uma contagem regressiva, a declaração **STEP** existente no laço **FOR... NEXT** deve ser seguida de um número negativo, destinado a diminuir sua freqüência.

PINTANDO O SETE E OUTROS NÚMEROS

Apenas por divertimento — e valiosa experiência em programação — você pode usar laços **FOR... NEXT** para criar variedades de efeitos gráficos. Aqui está uma amostra:

```
7 CLS 0
10 FOR N=0 TO 63
20 LET M=RND(32)-1
30 LET C=RND(9)-1
40 SET (N,M,C)
50 NEXT N
60 GOTO 10
```

```
10 FOR n=0 TO 21
20 LET m=RND*31
30 INK RND*7+1
40 PRINT AT n,m;"■"
50 NEXT n
60 GOTO 10
```

```
10 SCREEN 3
20 R=RND(-TIME)
30 FOR N=0 TO 191 STEP 4
40 LET M=RND(1)*255
50 LET C=RND(1)*15
60 PSET(M,N),C
70 NEXT N
80 GOTO 30
```

```
5  GR
10  FOR N = 0 TO 39
20  LET M =  RND (1) * 40
30  COLOR=  RND (1) * 15
40  PLOT M,N
50  NEXT N
60  GOTO 10
```

A linha 10 define a altura do desenho que será colocado no vídeo e diz ao computador para imprimi-lo linha por linha.

A linha 20 define a largura e, somada à linha 40, ordena ao computador que imprima pequenos quadrados aleatoriamente naquela largura disponível.

A linha 30 escolhe, ao acaso, as cores dos quadradinhos.

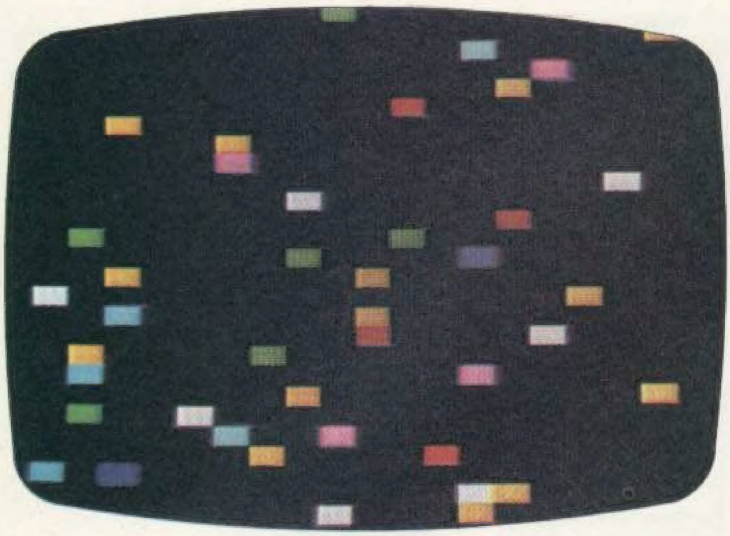
P&R

Os comandos ou palavras-chave em BASIC, tal como PRINT, GOTO, STEP, etc., precisam ser digitados sempre em letras maiúsculas?

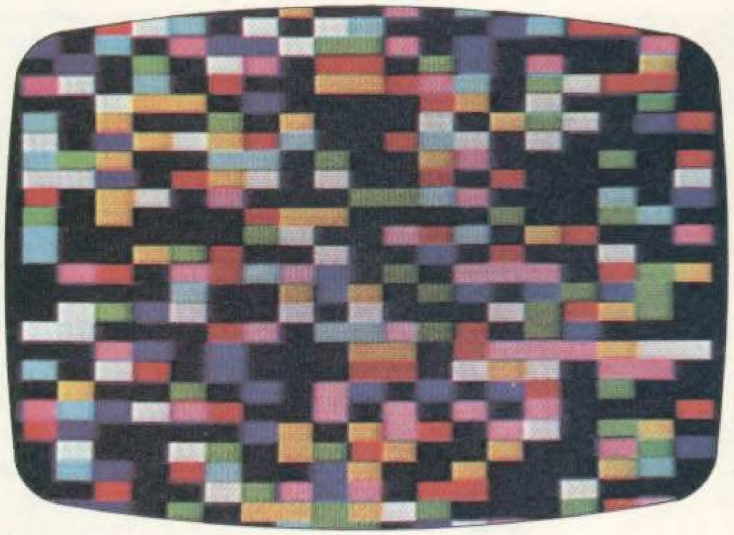
Sim, quase sempre. É preciso considerar, no entanto, as especificidades das diversas linhas. Alguns computadores, como os compatíveis com o ZX-81 e o Sinclair Spectrum não deixam margem a dúvidas, pois as palavras-chave são entradas por inteiro, em maiúsculas, ao se pressionar a tecla correspondente.

Outros computadores, como os compatíveis com a linha TK-2000 e os Apple II e Apple II Plus, não têm minúsculas (e nem aceitam comandos que não sejam em maiúsculas, caso você tenha essa opção em sua máquina).

Finalmente, os compatíveis com as linhas MSX e TRS-80 aceitam comandos em minúsculas, mas convertem internamente para maiúsculas todos os comandos em BASIC.



Inicialmente, forma-se na tela uma "colcha de retalhos"...



... que continua a se repetir. Esta é a versão para o TK-90X.

VARIAÇÕES SOBRE UM MESMO TEMA

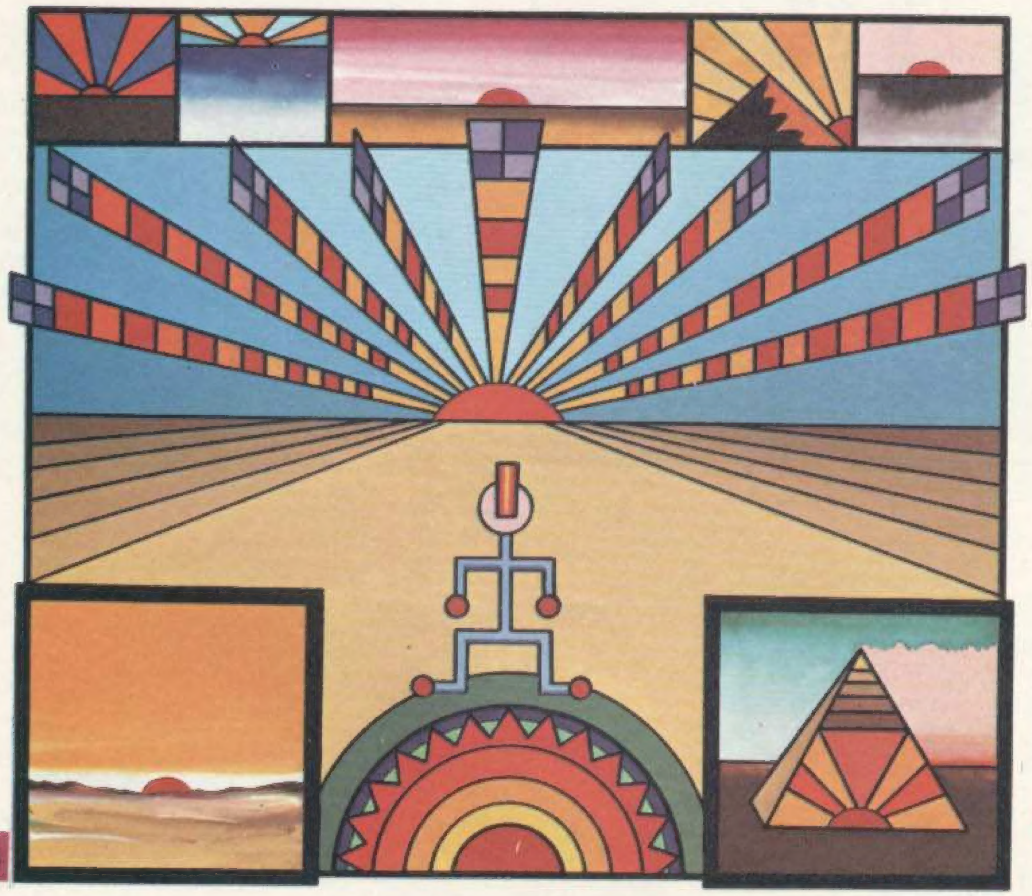
Experimentar variações desse tema de elaboração de gráficos vai ajudá-lo a se familiarizar com laços **FOR... NEXT** e com a função **RND**.

Aqui temos mais duas para cada máquina. Não se esqueça de digitar o comando **NEW** entre elas, para evitar que os programas se misturem.

TT

O TRS-80 comum e seus compatíveis não selecionam a cor do ponto gráfico; por isso o programa não tem muita graça. Se você quiser ver como é, substitua na versão abaixo a linha 40 e retire a linha 30:

```
40 SET (N,M)
```



```
8 CLS0
10 FOR N=1 TO 60
20 FOR M=0 TO 31
30 LET C=RND(9)-1
40 SET (N,M,C)
45 NEXT M
50 NEXT N
60 GOTO 10
```

Tente também a seguinte variação para o TRS-Color:

```
8 CLS0
10 LET N=RND(60)
20 FOR M=0 TO 31
30 LET C=RND(9)-1
40 SET (N,M,C)
45 NEXT M
60 GOTO 10
```

SS

```
10 FOR n=0 TO 21
20 FOR m=0 TO 31
30 INK RND*7+1
40 PRINT AT n,m;"■"
45 NEXT m
50 NEXT n
60 GOTO 10
```

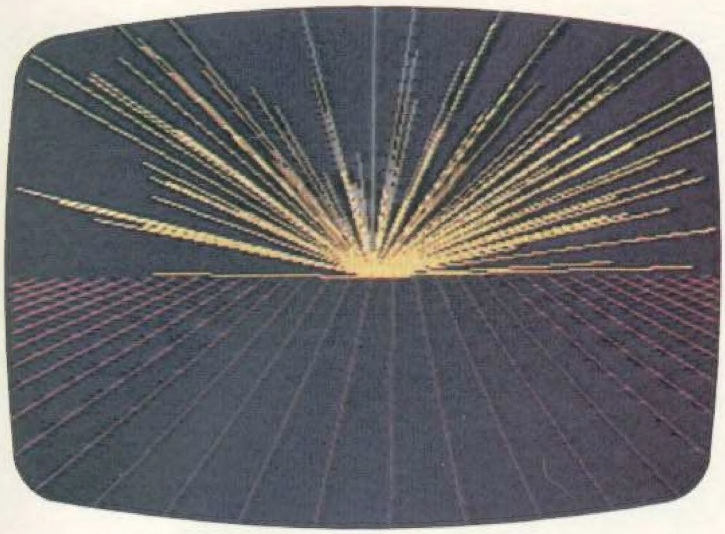
Da mesma forma, o ZX-81 não tem cor. Para rodar o programa, retire a linha 30 e use apenas letras minúsculas, quando digitá-lo.

Para o TK-90X, tente ainda a seguinte variação do programa. O que acontece de diferente?

```
10 LET n=RND*21
20 FOR m=0 TO 31
30 INK RND*7+1
40 PRINT AT n,m;"■"
45 NEXT m
69 GOTO 10
```

MX

```
10 SCREEN 3
```



Pôr-do-sol, de autoria do laço *FOR... NEXT* para o TK-2000.

A versão do Apple II para o Caleidoscópio.

```
20 R=RND(-TIME)
30 FOR N=0 TO 191 STEP 4
40 FOR M=0 TO 255 STEP 4
50 LET C=RND(1)*15
60 PSET(M,N),C
65 NEXT M
70 NEXT N
80 GOTO 30
```

```
10  GR
20  LET N =  RND (1) * 40
30  FOR M = 0 TO 39
40  COLOR=  RND (1) * 15
50  PLOT M,N
60  NEXT M
80  GOTO 20
```

Antes de ir adiante, aqui está uma pequena experiência que você deveria tentar — apenas no TK90X e no CP400. Apague a linha 45 dos dois primeiros programas e rode-os novamente com a seguinte modificação:

```
55 NEXT M
```

Você descobriu mais uma característica importante dos laços **FOR... NEXT**: quando se tem dois ou mais laços no mesmo programa, estes devem ficar ou "aninhados" um no outro ou separados. Se eles se cruzam, o programa não funciona.

PÔR-DO-SOL

Este programa usa um laço **FOR... NEXT** para criar um padrão de "pôr-do-sol". A primeira parte do programa define um ponto no meio da tela. Em seguida, ele desenha linhas luminosas que irradiam desse foco.

A segunda parte desenha uma série de linhas na metade inferior do vídeo, partindo de pontos fixos no "horizonte". Isso cria um belo efeito de perspectiva que pode ser aproveitado em programas futuros.

```
20 PMODE 3,1
30 PCLS 3
40 COLOR 2
50 SCREEN 1,0
60 FOR N=1 TO 80
70 LINE(127,95)-(256-RND(256),9
6-RND(96)),PSET
80 NEXT N
90 COLOR 4
100 FOR N=95 TO 191 STEP 12
110 LINE (127,95)-(0,N),PSET
120 LINE (127,95)-(255,N),PSET
130 NEXT N
140 FOR N=0 TO 255 STEP 10
150 LINE(127,95)-(N,191),PSET
160 NEXT N
170 GOTO 170
```

```
 5 BORDER 0: PAPER 0: INK 6:
 CLS
10 FOR n=1 TO 80
20 PLOT 127,75
30 DRAW INT (RND*250)-125,INT
 (RND*97)
40 NEXT n
45 INK 5
50 FOR n=75 TO 0 STEP -15
60 PLOT 127,75
70 DRAW -127,-n: PLOT 127,75:
 DRAW 127,-n
80 NEXT n
100 FOR n=-127 TO 127 STEP 20
110 PLOT 127,75
120 DRAW n,-75
130 NEXT n
```

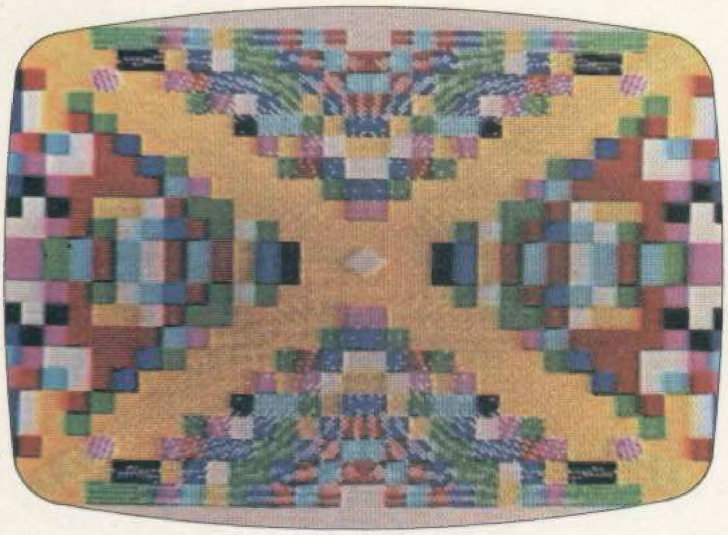
```
20 SCREEN 2
30 CLS
40 COLOR 11
50 R=RND(-TIME)
60 FOR N=1 TO 80
70 LINE (127,95)-(256-RND(1)*25
6,96-RND(1)*96)
80 NEXT N
90 COLOR 6
100 FOR N=95 TO 191 STEP 12
110 LINE (127,95)-(0,N)
120 LINE (127,95)-(255,N)
130 NEXT N
140 FOR N=0 TO 255 STEP 10
150 LINE (127,95)-(N,191)
160 NEXT N
170 GOTO 170
```

```
10  HGR
20  HCOLOR= 5
30  FOR I = 1 TO 80
40  HPLOT 139,79 TO  RND (1) *
279, RND (1) * 79
50  NEXT I
60  HCOLOR= 6
70  FOR I = 79 TO 159 STEP 10
80  HPLOT 139,79 TO 0,I
90  HPLOT 139,79 TO 279,I
100  NEXT I
110  FOR I = 0 TO 279 STEP 20
120  HPLOT 139,79 TO I,159
130  NEXT I
```

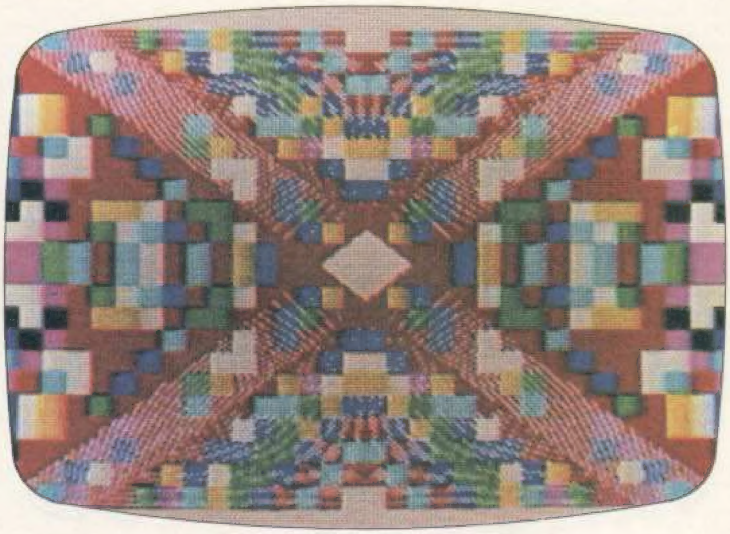
CALEIDOSCÓPIOS

Finalmente, temos um programa espetacular que funciona em quatro de nossas máquinas.

```
3 PMODE 3,1
6 PCLS
```



No TK-80X o Caleidoscópio é mais grosseiro e intenso



... mas seu padrão de formas e cores é sempre cambiante.

```
9 SCREEN 1,0
10 FOR L=0 TO 255 STEP 2
15 COLOR RND(4)
20 LINE(0,0)-(L,191),PSET
30 LINE (255,0)-(255-L,191),PSE
T
40 LINE(0,191)-(L,0),PSET
50 LINE (255,191)-(255-L,0),PSE
T
60 NEXT L
70 GOTO 10
```

```
10 FOR n=0 TO 255 STEP 3
15 INK RND*8
20 PLOT 0,0: DRAW n,175
30 PLOT 255,0: DRAW -n,175
40 PLOT 0,175: DRAW n,-175
50 PLOT 255,175: DRAW -n,-175
60 NEXT n
70 GOTO 10
```

```
3 R=RND(-TIME)
6 CLS
9 SCREEN 2
10 FOR L=0 TO 255 STEP 2
15 COLOR RND(1)*15
20 LINE (0,0)-(L,191)
30 LINE (255,0)-(255-L,191)
40 LINE (0,191)-(L,0)
50 LINE (255,191)-(255-L,0)
60 NEXT L
70 GOTO 10
```

```
10  HGR
20  FOR N = 0 TO 279 STEP 5
30  HCOLOR=  RND (1) * 6 + 1
40  HPLOT 0,0 TO N,159
50  HPLOT 279,0 TO 279 - N,159
60  HPLOT 0,159 TO N,0
70  HPLOT 279,159 TO 279 - N,0
80  NEXT N
90  GOTO 20
```

Cada um dos quatro segmentos desses padrões começa com um ponto em um canto do vídeo. O que o laço **FOR... NEXT** faz é contar até o outro lado do vídeo, enquanto um padrão de linhas é desenhado entre os dois pontos criados dessa forma.

A explicação exata de como os gráficos funcionam está em um capítulo posterior, mas apague as linhas 30 e 40 e você terá uma idéia geral do que acontece.

Você pode experimentar, ainda, digitando algumas das linhas responsáveis pelo desenho, variando as cores, mudando os **STEP** na linha 10 e excluindo o **GOTO** na linha final. Em minutos você pode criar centenas de padrões diferentes, um verdadeiro caleidoscópio de formas e cores.

Ao fazer tudo isso, você não estará só criando belas imagens no vídeo. Você estará também se familiarizando com uma das mais úteis "ferramentas" do programador.

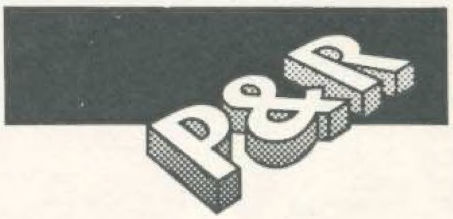
MICRO DICAS

USE O LAÇO CERTO

Existem muitas oportunidades para usar os comandos **FOR... NEXT** em laços, mas também muitas ocasiões em que você não deve empregá-los.

Quando você quiser, por exemplo, forçar o programa a realizar um número fixo de repetições, sem interrupção em quaisquer fases do processamento, use um laço **FOR... NEXT**.

Em contrapartida, se você quiser que uma seqüência de programa seja executada até que uma determinada condição seja atingida, para em seguida "sair" de dentro do laço, use uma declaração diferente, no lugar de **FOR... NEXT**. O comando mais recomendável em casos assim é o **GOTO**, em BASIC.

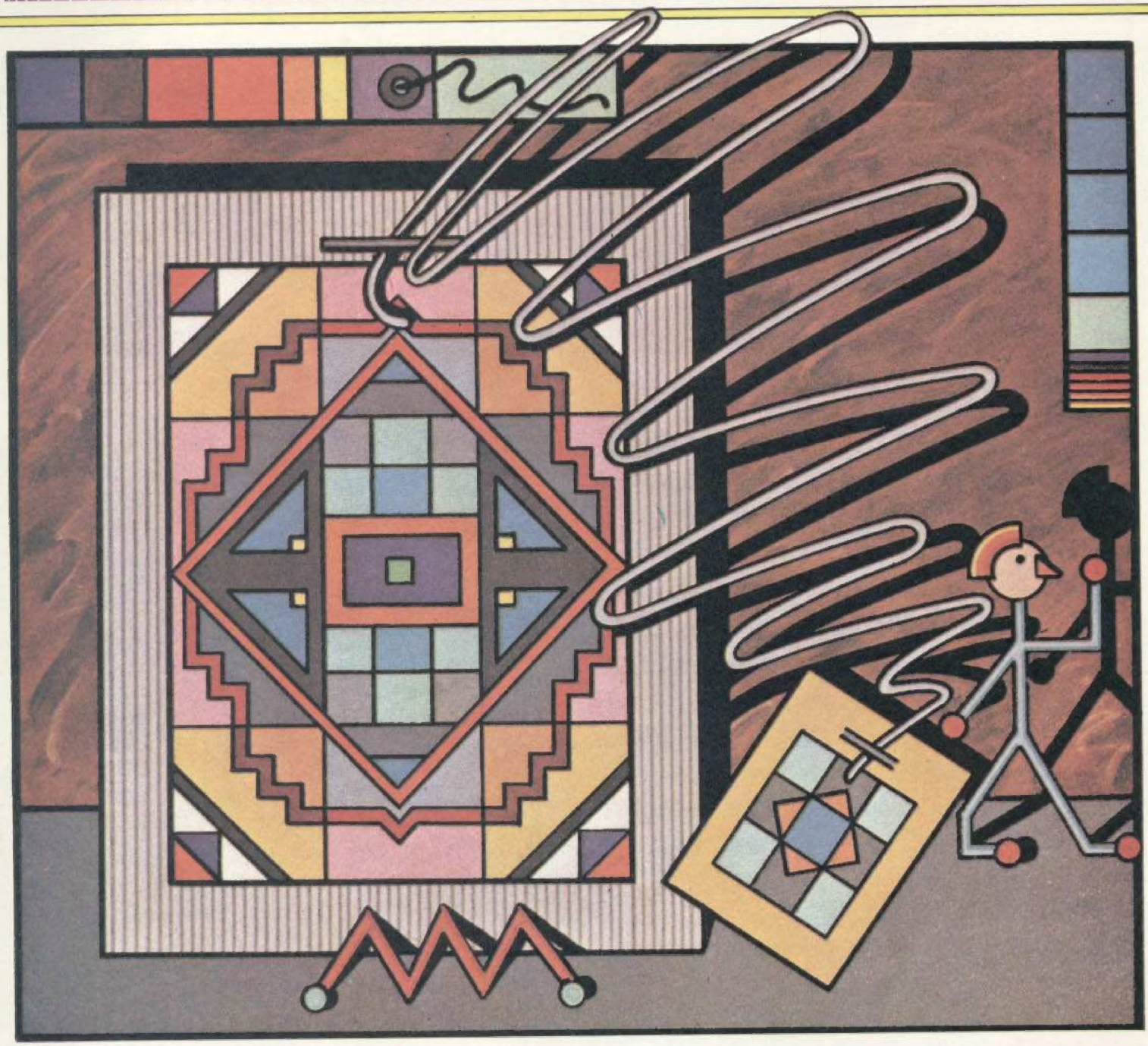


Como devo proceder para manter um registro de todas as variáveis usadas em meus programas, sem me perder entre tantos Xs e Ys?

Uma forma de evitar o uso desordenado de variáveis em um programa é dar-lhes nomes que lembrem o que elas representam (nomes mnemônicos). Por exemplo, se uma variável é usada para armazenar o número de espaçonaves abatidas em um jogo, podemos "batizá-la" de **NAVES**.

É fácil manter um registro de variáveis num programa curto. No caso de programas mais longos, convém explicitar mais claramente certas funções. Assim, utilize **FOR LINHA** = ... e **FOR COLUNA** = ... (se você estiver colocando coisas na tela) em vez de **FOR X** = ... e **FOR Y** = ...

Alguns computadores, como o TRS-80 e o Apple II, reconhecem apenas as duas primeiras letras de uma variável, descartando as demais. Neste caso, tente utilizar iniciais que funcionem como abreviaturas adequadas, como **T** para "tempo", **RC** para "recorde", **L** para "linha", e assim por diante.



COMO O COMPUTADOR É CAPAZ DE REPETIR TRECHOS DE UM PROGRAMA

A capacidade de repetir de forma controlada o mesmo trecho de um programa pode ser encontrada em todas as linguagens de programação. No BASIC, a programação de laços é muito facilitada pelos comandos **FOR, STEP** *e* **NEXT**, *explicados nesta lição; porém eles não são essenciais.*

É possível programar laços de repetição de outro modo, simplesmente usando-se uma variável contadora e uma linha com a declaração (ou comando) **IF**.

Dessa forma, a programação direta de repetições exemplifica de modo mais claro e objetivo a maneira que o computador usa, internamente, para repetir trechos de um programa.

Suponhamos que o programador queira imprimir dez vezes a mesma mensagem na tela, usando o comando **PRINT**. *Inicialmente, é preciso definir e igualar a 0 uma variável contadora (por exemplo,* **N**). *Em seguida, a cada repetição do laço, essa variável é incrementada, somando-se o número 1 ao valor anterior (este número corresponde, no caso, ao valor* **STEP** *no comando* **FOR**). *Finalmente, uma linha adicional do programa deve testar se esse valor de* **N** *é maior do que 10. Se não for, o laço será repetido por meio de um comando* **GO-TO** *e enviará o programa para trás. Se for maior do que 10, o programa terminará ou prosseguirá para outro trecho.*

APONTAR ...FOGO!

Jogos de ação dependem da capacidade do jogador em controlar os movimentos na tela. Veja como fazer isso e aprenda a disparar mísseis, integrando tudo num jogo divertido.

Os videogames para microcomputadores, como o *Space Invaders*, não teriam nenhuma graça se a movimentação e os disparos do canhão laser não pudessem ser controlados pelo jogador simultaneamente com o desenrolar do jogo. Esse tipo de controle através do teclado ou do *joystick* é uma característica de todos os jogos de ação, mesmo os mais simples.

O elemento mais importante dessa técnica de programação é como fazer para que o computador perceba automaticamente quando uma tecla qualquer é pressionada, sem que seja necessário acionar todas as vezes as teclas <**ENTER**> ou <**RETURN**>. Feito isso, o programa pode tomar decisões sobre que gráfico animar, que "disco voador" destruir, etc. Outro aspecto essencial é que, enquanto o jogador não acionar os controles de jogo, o computador poderá se dedicar à movimentação contínua de gráficos e figuras sobre a tela, usando as técnicas aprendidas na última lição de programação de jogos.

COMO DETECTAR TECLAS PRESSIONADAS

Em princípio, todos os micros usam o mesmo método para detectar se uma tecla foi pressionada. Mas, como os comandos do BASIC para essa função não foram padronizados, há uma grande variabilidade entre os diversos tipos de computadores.

TRS-80

Para a maioria dos computadores que utilizam o interpretador da linguagem Microsoft BASIC (adotada nos micros da linha TRS-80, Sinclair e MSX), a função utilizada para detectar se uma tecla foi pressionada chama-se **INKEY$**. Ao encontrar essa função em um programa, o computador executa uma "varredura" do teclado, para ver se alguma passou do estado de desligada para ligada, e retorna o caractere ao programa através da função **INKEY$**. Se nada foi pressionado, a função volta com um valor "vazio".

Por exemplo:

- APRENDA A DETECTAR SE UMA TECLA FOI PRESSIONADA
- DISPARE SEU MÍSSIL
- COMO CONTROLAR UMA FIGURA EM MOVIMENTO
- CONSTRUA BLOCOS GRÁFICOS
- DESTRUA O INIMIGO
- BASES DE MÍSSEIS
- A AUTO-REPETIÇÃO
- COMO MONTAR ALVOS MÓVEIS

Execute o programa e pressione qualquer tecla, à exceção de <**CAPS SHIFT**> ou <**SYMBOL SHIFT 1**> (nos Sinclair), <**BREAK**> ou <**SHIFT**> (nos TRS-80) ou <**CTRL STOP**> ou <**SHIFT**> (no MSX). O computador vai exibir um "grito de dor" no meio da tela. O programa funciona da seguinte maneira:

A linha 20 limpa a tela. A linha 30 faz com que o computador espere até que uma tecla seja acionada para continuar o programa. Note que não há espaços entre as aspas. Por causa disto, a linha 30 quer dizer: "Se **INKEY$** = nada, ou se nenhuma tecla está sendo pressionada, cheque novamente". É muito importante ter o **IF...THEN GOTO 30** porque de outra forma o computador verificaria apenas uma vez o teclado e só por uma fração de segundo.

Assim que uma tecla é acionada, **INKEY$** é igualado à tecla. Por exemplo, se "3" é pressionado, então **INKEY$** = "3". Isso basta para que a linha 40 imprima "Ai!" na tela.

Na maioria dos jogos você deve pressionar uma tecla para mover um tanque, uma míssil, etc. Se você mudar a linha 40, verá como isto é feito (no TK-85 use um "D" maiúsculo):

```
20 CLS
30 LET A$=INKEY$:IF A$="" THEN
GOTO 30
40 PRINT @269,"OPA!"
```

```
20 CLS
30 IF INKEY$="" THEN  GOTO 30
40 PRINT AT 11,14;"Ai!"
```

```
20 CLS
30 LET A$=INKEY$:IF A$="" THEN
GOTO 30
40 LOCATE 18,11:PRINT "Ui !"
```

```
40 IF A$="D" THEN PRINT @264,"H
UM, ISSO E BOM!" ELSE PRINT @26
9,"OPA!"
```

```
40 IF INKEY$="d" THEN  PRINT
AT 11,9;"Hum, isso e bom!":
STOP
50 PRINT AT 11,14;"Ai!"
```

```
20 CLS
30 LET A$=INKEY$:IF A$="" THEN
GOTO 30
40 IF A$="D" OR A$="d" THEN LOC
ATE 10,11:PRINT "mmm... isso é
bom !":END
50 LOCATE 18,11:PRINT "Ui !"
```

A linha 40 agora verifica se a tecla "D" foi pressionada, ou seja, se **INKEY$** é igual a "D" ou a "d". Se não for, o programa para os micros da linha Sinclair, que não têm a declaração **ELSE**, vão ignorar a linha 40 e passar para a linha 50. Qualquer outra tecla diferente de "D" mostrará, quando pressionada, a segunda mensagem de "reação emocional". Nos micros de outras linhas, o **IF...THEN...ELSE** permite programar em uma única linha as duas alternativas.

Existem mais duas exigências importantes nesse programa. A primeira é que o "D" ou "d" precisa estar entre aspas, ou o computador o interpretará como uma variável. A segunda vale apenas para o TK-90X: nessa linha de micros, deve-se usar um "d" minúsculo; caso contrário você deve pressionar <**CAPS SHIFT**> e "d" para que o programa funcione.

Tanto nesse programa quanto no anterior é possível notar que todos os computadores utilizam exatamente a mesma técnica de investigação do teclado. Apenas a maneira de colocar uma mensagem no meio da tela é diferente para cada um.

DISPARE SEU MÍSSIL

Agora você verá que a partir da detecção do pressionamento de uma tecla usando o **IF INKEY$** = "tecla" é muito fácil disparar um míssil ou mover uma base de foguetes. Este programa dispara um míssil de uma base quando a tecla "F" é acionada:

```
20 CLS
30 PRINT @397,"▮▮▮"
40 LET K$=INKEY$:IF K$=""THEN G
OTO 40
50 IF K$="F" THEN LET M=334 ELS
E GOTO 40
60 PRINT @M,"!"
70 PRINT @M+32," "
80 LET M=M-32
90 IF M>0 THEN GOTO 60
100 GOTO 20
```

```
10 CLS
20 PRINT @797,"▮▮▮"
30 K$=INKEY$ : IF K$="" GOTO 30
```

```
40 IF  K="F"   THEN M=374  ELSE
   GOTO 30
50 PRINT @M,CHR$(94);
60 PRINT @M+64," ";
70 M=M-64
80 IF M>0 GOTO 50
90 GOTO 10
```

Para rodar também no ZX-81, digite o programa abaixo apenas com letras maiúsculas:

```
20 CLS
30 PRINT AT 21,14;"▟▙"
40 IF INKEY$="" THEN  GOTO 40
50 IF INKEY$<>"f" THEN  GOTO 40
55 LET y=20
60 PRINT AT y,15;"↑"
70 LET y=y-1
75 PAUSE 1
80 PRINT AT y+1,15;" "
90 IF y>0 THEN  GOTO 60
```

```
20 CLS
30 LOCATE 17,20:PRINT "▟▙"
40 LET K$=INKEY$:IF K$="" THEN
GOTO 40
50 IF K$="F" OR K$="f" THEN LET
 M=20 ELSE GOTO 40
60 LOCATE 18,M:PRINT" "
70 LOCATE 18,M+1:PRINT " "
80 LET M=M-1
90 IF M>0 THEN GOTO 60
100 GOTO 20
```

A linha 30 imprime uma base de mísseis formada de três símbolos (no Sinclair e no MSX, três símbolos gráficos), na parte mais baixa da tela (última linha).

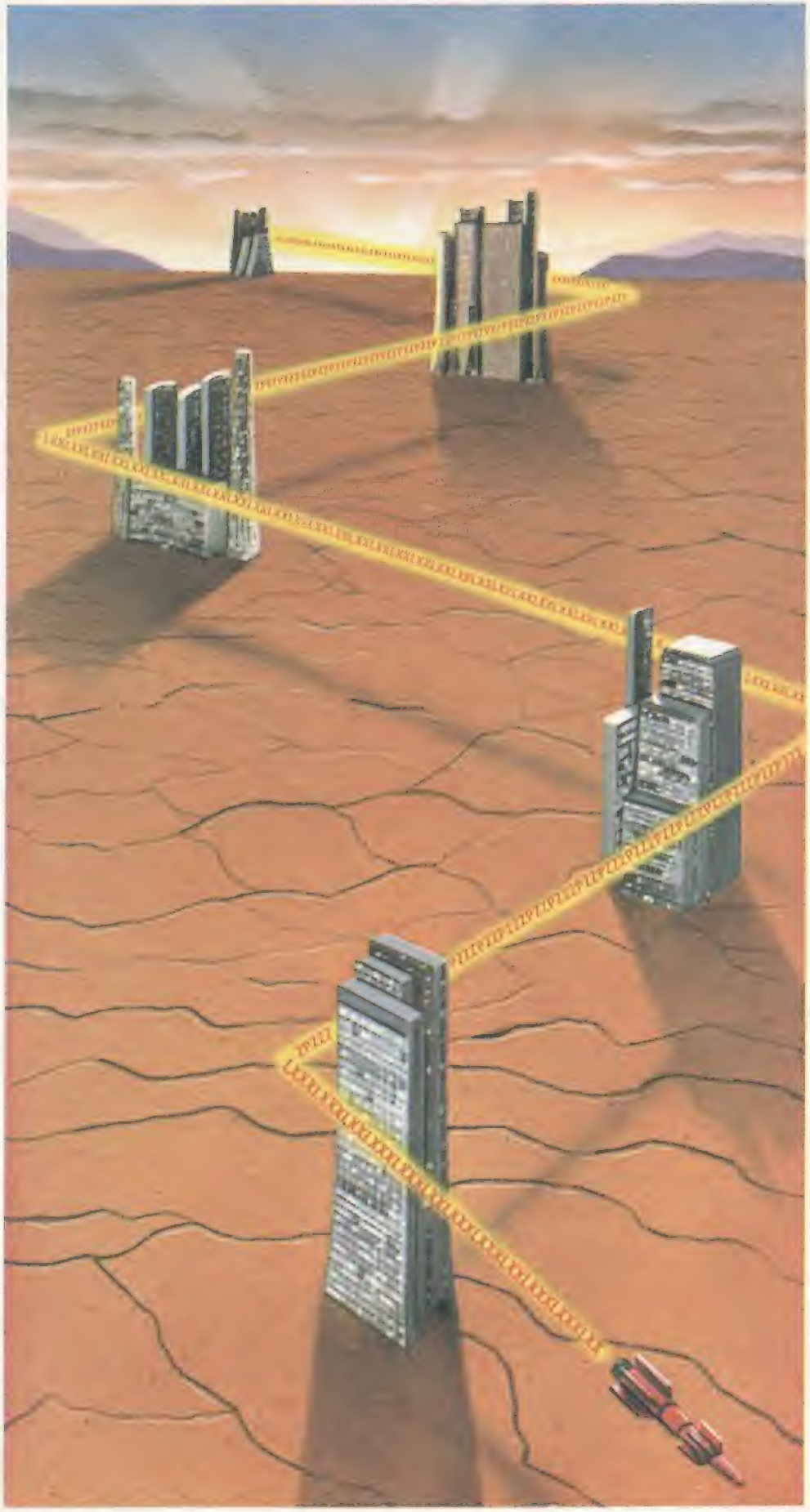
Na linha 40, **LET K$=INKEY$** é muito importante, visto que você quer checar se a tecla foi pressionada várias vezes durante o programa. Esta não funcionaria sem essa declaração. O computador se recorda do valor de **INKEY$** apenas por uma fração de segundo e, se você não checar **INKEY$** suficientemente rápido, ele pode esquecer que a tecla foi pressionada.

Mas, você pode também fazer com que ele se lembre do que foi teclado dando um nome, **K$**, à tecla que foi pressionada e checando mais tarde (versão para a linha TRS), ou dando um novo comando **INKEY$.**

A linha 50 verifica se "F" foi pressionado. Se não, o programa volta à linha 40 e continua a varrer o teclado. "M" é a posição do míssil depois de ter sido disparado.

A linha 60 imprime o míssil na tela e a linha 70 se encarrega de apagá-lo de suas posições anteriores.

A linha 80 faz o míssil se deslocar pa-



ra cima na tela, e a linha 90 evita que o computador tente imprimir o míssil numa posição fora do vídeo, o que provocaria uma mensagem de erro. A primeira posição de tela tem o número 0, e o computador não pode imprimir nada em uma posição de número menor que 0. Quando M = 0 o programa é reiniciado. A técnica a ser utilizada para fazer o míssil subir vai depender, agora, do formato da tela do computador utilizado.

A linha 80 subtrai um número constante N da posição anterior do míssil (variável **M**), de maneira que ele avança uma linha em direção ao topo da tela a cada vez que é impresso. Esse número vale 32 para os micros compatíveis com o TRS-Color, pois há 32 colunas na tela do computador. Já para o TRS-80, o número é 64.

A linha 80 subtrai 1 da posição anterior do míssil, pois o comando de posicionamento utilizado, o **PRINT AT**, identifica as coordenadas (linha, coluna) onde serão exibidos os gráficos. Assim, a coluna é mantida constante, para o míssil subir "reto", na coluna 15, e o que varia é Y, ou a altura dele (lembre-se de que Y = 0 está no topo da tela, e não embaixo.

O comando de posicionamento é o **LOCATE**, que dá as coordenadas para posicionamento. A linha 80 tem por função diminuir em 1 o valor anterior do míssil (variável **M**), para que ele se desloque para cima, na mesma coluna (constante 18).

MOVIMENTOS SOBRE A TELA

Da forma que está, o programa da base de mísseis é bastante limitado para um jogo de "artilharia", mas, imprimindo movimento lateral à base conseguiremos melhorá-lo um pouco. Vejamos como fazer isso:

```
20 CLS
30 LET P=397
40 PRINT @P,CHR$(143)+CHR$(140)
+CHR$(128)+CHR$(140)+CHR$(143)
50 LET K$=INKEY$:IF K$="" THEN
GOTO 50
60 IF K$="E" THEN LET P=P-1:GOT
O 90
70 IF K$="D" THEN LET P=P+1:GOT
O 90
80 GOTO 50
90 IF P<384 OR P>411 THEN GOTO
50
100 GOTO 40
```

A linha 30 determina a posição inicial da base, e a linha 40 coloca o míssil naquela posição.

A linha 50 verifica o teclado, fazendo o computador esperar até que uma tecla seja pressionada.

A linha 60 investiga se "L" foi pressionado. Se foi, ela move a base uma posição para a esquerda, subtraindo 1 do número que determina a posição da base. A linha 70 confere se "R" foi pressionado e nesse caso muda a posição da base adicionando 1 à coordenada. A linha 90, finalmente, verifica se o programa está dizendo ao computador para colocar a base fora da tela — em caso afirmativo, o programa retorna à linha 50.

```
20 CLS
30 P=797
40 PRINT@P,CHR$(32)+CHR$(184)+
   CHR$(191)+CHR$(180)+CHR$(32)
50 K$=INKEY$ : IF K$="" GOTO 50
60 IF K$="L" THEN P=P-1
70 IF K$="R" THEN P=P+1
80 GOTO 50
90 IF P<767 OR P>828 GOTO 50
100 GOTO 40
```

O programa para o TRS-80 funciona de modo idêntico ao programa para o TRS-Color. Note que os caracteres gráficos têm símbolos diferentes, bem como as coordenadas de posicionamento dos comandos **PRINT** @.

Para rodar também no ZX-81, digite o programa abaixo apenas com letras maiúsculas. Além disso, ponha os comandos **LET** da linha 80 em duas linhas separadas.

```
 30 CLS
 40 LET x=15
 50 LET y=13
 60 PRINT AT y,x;"■■■"
 70 IF INKEY$="" THEN  GOTO 70
 80 LET lx=x: LET ly=y
 90 PRINT AT ly,lx;"   "
100 IF INKEY$="q" THEN  STOP
110 IF INKEY$="w" THEN  LET y=
y-1
120 IF INKEY$="z" THEN  LET y=
y+1
130 IF INKEY$="a" THEN  LET x=
x-1
140 IF INKEY$="s" THEN  LET x=
x+1
150 IF x<1 OR x>29 THEN  LET x
=1x
160 IF y<1 OR y>20 THEN  LET y
=1y
170 GOTO 60
```

Posso escolher qualquer tecla para operar os controles de um jogo no microcomputador?

Sim. Tudo o que você tem a fazer é mudar os caracteres nas linhas do programa que testam qual foi a tecla pressionada e detectada por **INKEY$**. Mas cuidado: o que parece ser bastante lógico na teoria às vezes funciona mal na prática. Por exemplo, usar as teclas E, D, C, B para representar esquerda, direita, em cima e embaixo, dificulta as ações do jogador. Por isso, costumam-se usar as teclas de movimentação do cursor, principalmente nos computadores onde elas estão aglomeradas no mesmo ponto do teclado, ou presentes em pares complementares. Outra dica é usar A e Z, para movimentar para cima e para baixo, e K e L para direita e para esquerda, pois ocupam posições opostas no teclado. Já a barra de espaços é muito boa para efetuar disparos, pois o polegar pode se apoiar nela.

As linhas 40 e 50 estabelecem a posição inicial da base de mísseis, na 13ª linha da tela, e quinze espaços à esquerda. A linha 60 mostra a base na tela.

A linha 70 faz o computador esperar até que alguma tecla seja impressa. Se foi, a base é movida.

As linhas 80 e 90 "perseguem" a base de misseis em movimento, colocando três espaços em branco sobre sua última localização; de modo a apagá-la. Como a linha 60 está dentro do laço de repetição e sempre vem antes da linha 90, a base é sempre impressa em sua nova posição antes que a posição antiga seja apagada.

A linha 100 termina o programa se a letra Q (de *quit*, em inglês, que significa abandone) for pressionada.

A linha 110, por sua vez, verifica se a tecla "W" foi acionada, subtraindo 1 do valor de "Y" em caso afirmativo. O efeito disso é a movimentação da base uma linha para cima.

As linhas 120 a 140 operam de forma similar. A linha 120 move a base para baixo se "Z" é pressionado.

A linha 130, move para a esquerda,

caso "A" seja acionado, e a 140 para a direita, ao se pressionar "S". A linha 150 evita que a base seja colocada fora das laterais da tela.

A linha 160 impede que a base saia por cima ou por baixo da tela. Neste caso, fazer **Y = LY** resolve o problema. A linha 170 fecha o laço, reiniciando todo o processo.

```
30 CLS
40 LET X=17
50 LET Y=20
60 LOCATE X,Y:PRINT "▟▙"
70 A$=INKEY$:IF A$="" THEN GOTO
 70
80 LET LX=X:LET LY=Y
90 LOCATE LX,LY:PRINT"   "
100 IF A$=CHR$(27) THEN END
110 IF A$=CHR$(28) THEN X=X+1
120 IF A$=CHR$(29) THEN X=X-1
130 IF A$=CHR$(30) THEN Y=Y-1
140 IF A$=CHR$(31) THEN Y=Y+1
150 IF X<0 OR X>36 THEN X=LX
160 IF Y<0 OR Y>20 THEN Y=LY
170 GOTO 60
```

O programa para a linha MSX é bastante compacto e elegante. A linha 60 coloca a base em posição inicial (três caracteres gráficos), que é definida nas linhas 40 e 50, ou seja, na linha 20 e na coluna 17 (no meio da última linha da tela). A linha 70 detecta se alguma tecla foi pressionada.

Se alguma tecla foi pressionada, o programa armazena em **LX** e **LY** as coordenadas da última posição da base (X e Y), para usá-las posteriormente (linha 80), e apaga a base nessa posição, por meio da linha 90.

As linhas 100 a 140 detectam se a tecla apertada foi uma das seguintes: flecha para cima, flecha para baixo, flecha para esquerda e flexa para direita (teclas de controle do cursor), que são identificadas pelos seus códigos respectivos (29, 28, 30 e 31). O jogo termina quando a tecla **CLEAR** (código 26) tiver sido pressionada.

As linhas 150 e 160 verificam se as bordas da tela foram ultrapassadas. Se foram, retornam à posição da base para as últimas coordenadas (**LX** e **LY**).

Finalmente, a linha 60 faz o ciclo se repetir, para a próxima pressão à tecla.

Por que o meu programa "bomba" (isto é, falha) se, num jogo qualquer, a figura que está sendo movimentada atinge a borda da tela?

Provavelmente porque um dos valores que controlam o posicionamento da impressão do gráfico na tela tornou-se muito grande ou muito pequeno, excedendo os limites da tela. Portanto, verifique de novo as linhas que fazem essa tarefa, caso você obtenha alguma mensagem de erro do computador, neste sentido. Por precaução, é sempre recomendável testar os valores de posicionamento antes de imprimir alguma coisa na tela: se eles excederem os limites, podem ser corrigidos para ficarem dentro da tela.

FAÇA O SEU JOGO

Agora você já tem alguns blocos a partir dos quais podem ser montados vários jogos. O exemplo abaixo mostra como usá-los para desenvolver um jogo bem simples, em que um canhão móvel dispara mísseis em direção a um alvo (uma estrela):

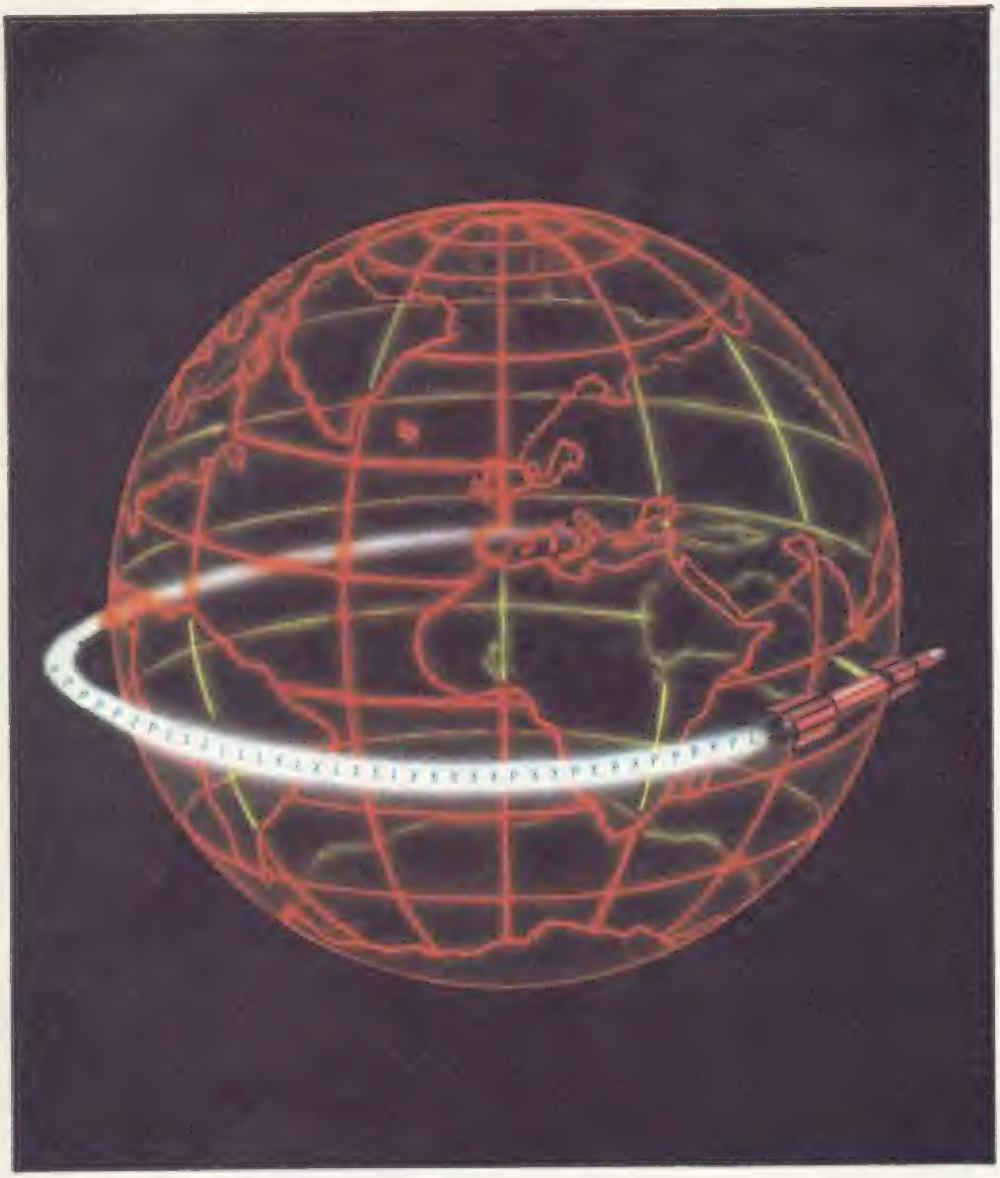
```
20 CLS
30 FOR N=1 TO 100:NEXT N
40 LET PO=430
50 LET B$=CHR$(143)+CHR$(140)+C
HR$(128)+CHR$(140)+CHR$(143)
60 LET A=RND(30)+64
70 PRINT @A,"*"
80 LET LP=PO
90 PRINT @PO,B$
100 IF PEEK(340)=247 THEN LET P
O=PO-1
110 IF PEEK(338)=247 THEN LET P
O=PO+1
120 IF PO<415 OR PO>444 THEN LE
T PO=LP
130 LET K$=INKEY$
140 IF K$="F" THEN LET M=PO-30
ELSE GOTO 80
150 PRINT @M,"↑";
160 PRINT @M+32," ";
170 LET M=M-32
180 IF M=A THEN GOTO 20
190 IF M>0 THEN GOTO 150 ELSE P
RINT @M+32," ";
200 GOTO 80
```

```
20 CLS
30 FOR N=1 TO 100:NEXT N
40 PO=800
50 B$=CHR$(32)+CHR$(184)      +
   CHR$(191)+CHR$(180)        +
   CHR$(32)
60 A=RND(60)+2
70 PRINT@A,"*";
80 LP=PO
90 PRINT@PO,B$;
100 IF PEEK(14344)<>0 THEN  PO=
    PO-1
110 IF PEEK(14368)<>0 THEN  PO=
    PO+1
120 IF  PO<767  OR PO>828  THEN
    PO=LP
130 K$=INKEY$
140 IF K$="F" THEN M=PO-62 ELSE
    GOTO 80
150 PRINT@M,CHR$(94);
160 PRINT@M+64," ";
170 M=M-64
180 IF M=A GOTO 20
190 IF  M>0 THEN GOTO 150  ELSE
    PRINT @M+64," ";
200 GOTO 80
```

Para rodar também no ZX-81, digite o programa abaixo apenas com letras maiúsculas:

```
 20 CLS
 30 PAUSE 25
 40 LET x=15: LET y=21
 50 LET B$=" ▟▙ "
 60 LET a=INT (RND*28)+2
 70 PRINT AT 2,a;"*"
 80 LET xx=x
 90 PRINT AT y,x;B$
100 IF INKEY$="z" THEN  LET x=
x-1
110 IF INKEY$="x" THEN  LET x=
x+1
120 IF x<0 OR x>27 THEN  LET x
=xx
140 IF INKEY$<>"f" THEN  GOTO
80
145 LET m=y-1
150 PRINT AT m,x+2;"↑"
160 PRINT AT m+1,x+2;" "
170 LET m=m-1
180 IF m=2 AND x+2=a THEN
GOTO 20
190 IF m<>1 THEN  GOTO 150
195 PRINT AT m+1,x+2;" "
200 GOTO 80
```

```
20 CLS
40 LET X=17:LET Y=20
50 LET B$=" ▟▙ "
60 LET A=INT(RND(1)*36)+2
70 LOCATE A,0:PRINT "*"
80 LET XX=X
90 LOCATE X,Y:PRINT B$
95 LET A$=INKEY$:IF A$="" THEN
GOTO 95
100 IF A$=CHR$(28) THEN LET X=X
+1
110 IF A$=CHR$(29) THEN LET X=X
-1
120 IF X<0 OR X>35 THEN LET X=X
X
130 IF A$<> " " THEN GOTO 80
140 LET M=Y-1
150 LOCATE X+2,M:PRINT "▀"
160 LOCATE X+2,M+1:PRINT " "
170 LET M=M-1
180 IF M=0 AND X+2=A THEN GOTO
20
```



```
190 IF M<>0 THEN GOTO 150
200 LOCATE X+2,M+1:PRINT" "
210 GOTO 80
```

Quando você executar o programa, verá uma estrela perto do topo da tela. As teclas Z e X movem a base de foguetes para a esquerda e a direita, e a tecla F dispara o míssil para destruir a estrela em sua posição.

Pense no programa como tendo três partes: uma, até a linha 70; outra, da 80 até a 120; e a última, da 130/140 até a 200.

As linhas que vão de 130/140 até 200 são similares ao programa anterior que dispara o míssil.

As variáveis e os **GOTO** foram mudados, mas a única linha adicionada foi a 180. Ela verifica se o míssil e a estrela estão no mesmo lugar. O programa recomeça em caso positivo.

A porção central, da linha 80 à 120, é uma versão compacta do programa "Movimentos Sobre a Tela". Os **PEEK** no programa para o CP400 verificam se Z ou X foram acionados e alteram **PO** de maneira apropriada.

A primeira parte do programa tem várias funções. A linha 30 provoca uma pequena pausa antes que o programa continue. Isso é importante quando a linha 180 fecha o laço no fim do programa. As linhas 40 e 50 determinam a posição inicial da base e definem sua forma.

MELHOR MOVIMENTAÇÃO

Pressionar a tecla que move a base para a direita ou para a esquerda toda vez que se precisa deslocar figuras é bastante cansativo. Então, usualmente montamos um mecanismo de auto-repetição.

Alguns micros, como os da linha MSX, possuem auto-repetição em todas as teclas; por isso o programa dado acima não necessita qualquer modificação: basta ficar pressionando a tecla de comando para baixo, que a base se move automaticamente.

Já para os computadores que não dispõem desses recursos, temos que nos satisfazer com o **INKEY$**, mesmo. Mas é difícil programar movimentos contínuos usando **INKEY$**.

Um modo de se resolver esse problema é ilustrado a seguir.

```
20 CLS
30 LET BL$=CHR$(128)
40 LET PO=238
50 PRINT @PO,BL$
60 LET LP=PO
70 IF PEEK(340)=247 THEN LET PO
=PO-1:GOTO 120
80 IF PEEK(338)=247 THEN LET PO
=PO+1:GOTO 140
90 IF PEEK(338)=251 THEN LET PO
=PO-32:GOTO 150
100 IF PEEK(342)=253 THEN LET P
O=PO+32:GOTO 150
110 GOTO 70
120 IF(LP AND 31)=0 THEN LET PO
=LP
130 GOTO 150
140 IF(PO AND 31)=0 THEN LET PO
=LP
150 IF PO>510 OR PO<0 THEN LET
PO=LP:GOTO 70
160 PRINT @LP," ";
170 PRINT @PO,BL$;
180 GOTO 60
```

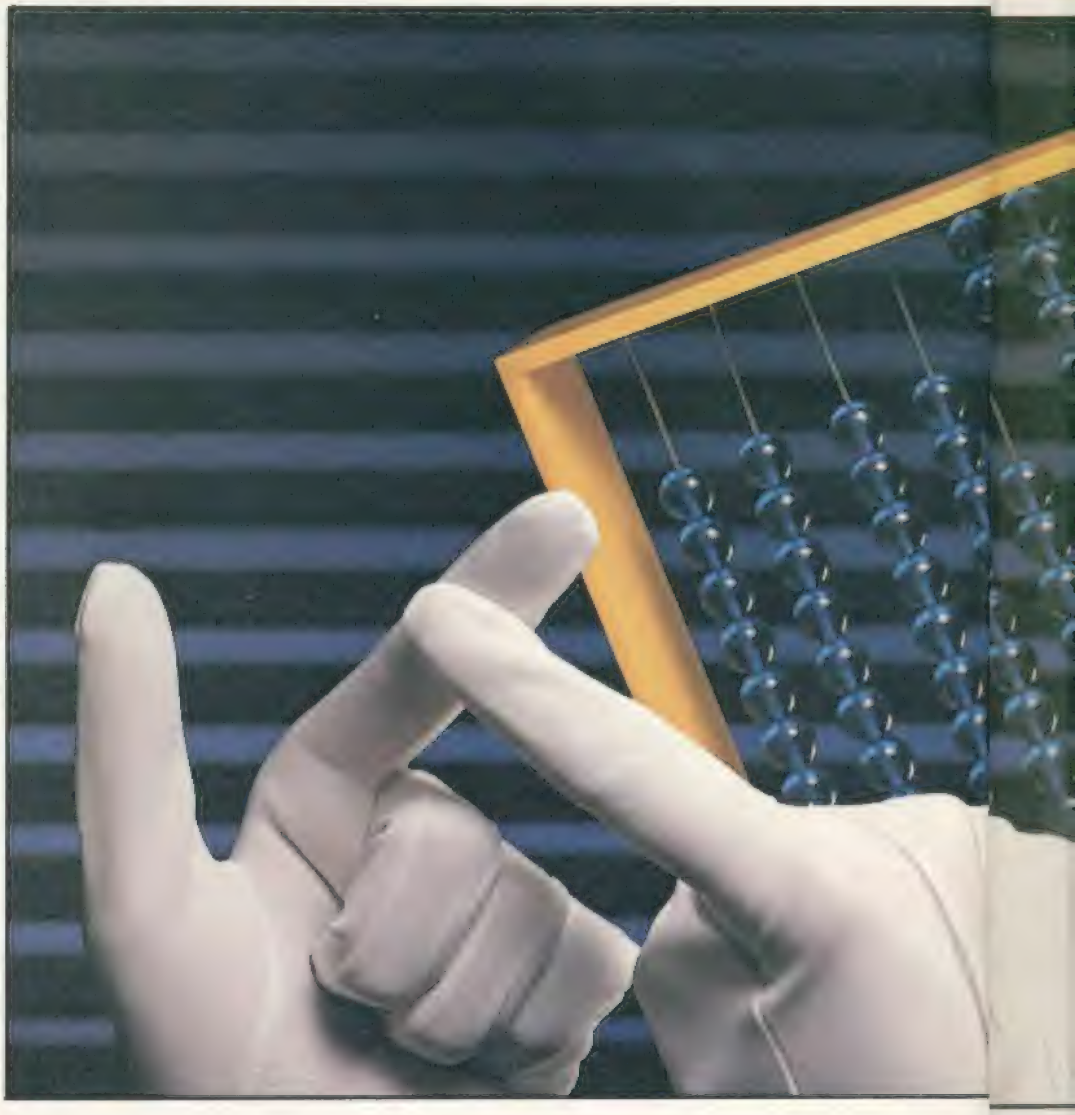
```
20 CLS
30 BL$=CHR$(191)
40 PO=540
50 PRINT@PO,BL$;
60 LP=PO
70 IF  PEEK(14344)<>0 THEN  PO=
   PO+1:GOTO I20
80 IF  PEEK(14368)<>0 THEN  PO=
   PO-1:GOTO 140
90 IF  PEEK(14352)<>0 THEN  PO=
   PO-64:GOTO 150
100 IF PEEK(14400)<>0 THEN  PO=
    PO+64:GOTO 150
110 GOTO 70
120 IF (LP AND 63)=0 THEN PO=LP
130 GOTO 150
140 IF (PO AND 63)=0 THEN PO=LP
150 IF PO>1022 PR PO<0 THEN PO=
    LP:GOTO 70
160 PRINT@LP," ";
170 PRINT@PO,BL$;
180 GOTO 60
```

Ao executar este programa você terá um bloco posicionado no centro da tela. O programa permite que você o mova para cima e para baixo, para a direita e para a esquerda.

APRENDA A CONTAR COM UM DEDO SÓ

Os computadores contam èm binário, como se tivessem milhões de mãos com um dedo só. Para aprender a programar em código de máquina, você deve conhecer esse sistema.

Um dos problemas de se aprender código de máquina é que você precisa entender um pouco da teoria dos números. Não é uma tarefa tão difícil quanto pode parecer. Se você sabe contar até 16, não terá qualquer dificuldade. Mas, inicialmente, é preciso aprender a contar até 2.



POR QUE A BASE 10?

Mesmo a pessoa mais avessa a matemática não tem qualquer dificuldade para dizer as horas ou acompanhar o resultado de uma partida de futebol. O uso dos números faz parte do dia-a-dia de tal forma que nunca nos preocupamos com a maneira com que eles funcionam.

No mundo ocidental usamos habitualmente um sistema numérico baseado no número 10. Isto significa que começamos uma contagem usando os números de 0 a 9. Se precisarmos de uma grandeza maior do que 9 (por exemplo, 10), usamos os dígitos disponíveis. Assim, colocamos o 1 na "casa" da esquerda e o 0 à sua direita.

Este é chamado um sistema de numeração de base 10 ou decimal, porque o valor do dígito é multiplicado por 10 a cada posição que contamos, a partir da direita. Por exemplo, o número 3275 vale $5 + 7\times10 + 2\times10\times10 + 3\times10\times10\times10$ ou seja $5+10+200+3000$. Cada dígito multiplica seu valor absoluto por 10 a cada vez que é movido uma posição mais à esquerda. Todo sistema que usa esse método para representar números é chamado de *posicional*.

Tudo isso parece óbvio porque fazemos esses cálculos todos os dias, sem pensar muito neles. Entretanto, existem outros sistemas de numeração, diferentes do decimal, como por exemplo, os que são usados nos modernos micros.

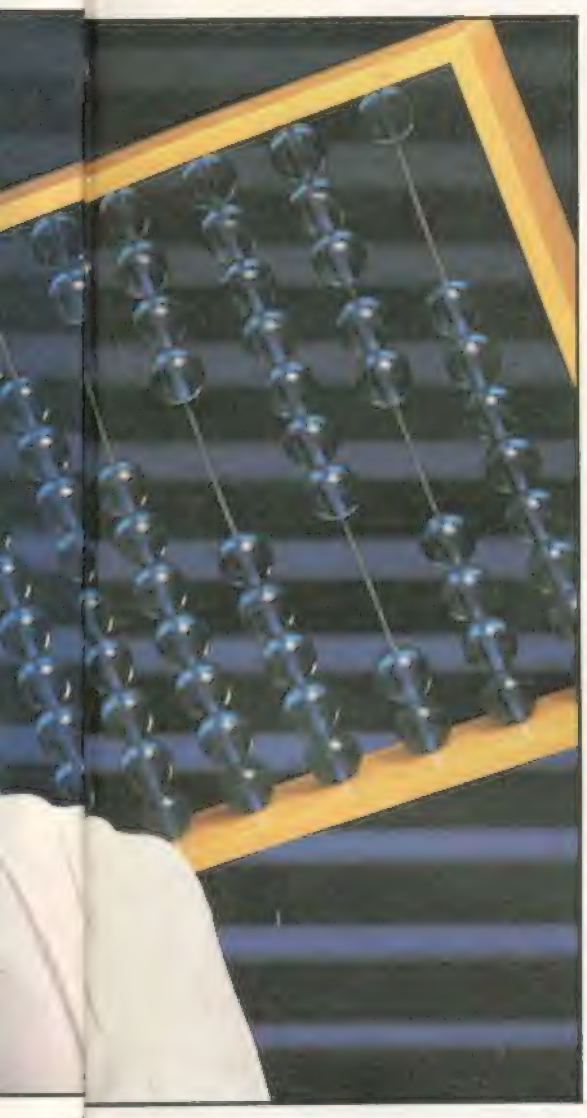
SISTEMAS ANTIGOS E NOVOS

Os antigos babilônios tinham um sistema de numeração baseado no número 60. Os vestígios disso podem ser vistos nas nossas medidas de tempo e de ângulos: existem 60 segundos em cada minuto, 60 minutos em cada hora, assim como 60 minutos de arco em um grau e 6 vezes 60 graus — ou seja, 360 — num círculo completo.

Você pode contar até 59 segundos, mas, se adicionar mais um segundo a esse número, terá um minuto e recomeçará a contar os segundos do 0. E quando tiver 59 minutos e 59 segundos, um segundo a mais fará uma hora, sendo zerados os minutos e segundos.

Resquícios de numeração em outras bases podem ser encontrados em velhos sistemas de medidas da Inglaterra. Existem 8 pintas em cada galão, 12 polegadas em cada pé e 16 onças em cada libra. Outra base que ocorre com freqüência é o número 12. Temos 12 polegadas em um pé, 12 unidades em uma dúzia e 12 dúzias em uma grosa. A base 12 era muito usada tanto para dinheiro como para bens, devido à facilidade com que se divide por 2, 3, 4 ou 6. Certas coisas tinham que ser divididas entre várias pessoas, e a base 12 tornava as transações muito mais simples. O número 10, ao contrário, só é divisível por 2 e 5. Vemos, portanto, que muitas vezes é conveniente usar outra base que

- O SISTEMA DECIMAL
- OUTROS SISTEMAS NUMÉRICOS
- COMO CONTAR EM BASE 9
- CONTAS EM DIFERENTES BASES
- CONVERSOR PARA TODAS AS BASES
- O SISTEMA BINÁRIO
- BITS E BYTES
- COMO O COMPUTADOR FAZ SOMAS



Os dez dedos das mãos levaram o homem a contar usando a base 10. Auxiliares de cálculo simples, como o ábaco, ou contador, são um modelo mecânico de nossos dedos.

não 10. Uma libra de peso, por exemplo, pode ser dividida em onças por sucessivas divisões ao meio.

CONTANDO COM NOVE DEDOS

Não há nenhuma razão que impeça o uso de um sistema baseado em qualquer número escolhido ao acaso. Se tivéssemos nascido, todos, com 1 dedo a menos em uma das mãos, provavelmente usaríamos um sistema "nonário" ou seja, de base 9. Contaríamos normalmente até 8 e, para obter o número seguinte, colocaríamos o 1 numa posição mais à esquerda e o 0 na posição à direita. Ou seja, o decimal 9 seria representado no sistema nonário pelo número 10.

Da mesma forma, se somássemos 1 a 18, teríamos 20; e 100 se somássemos 1 a 88.

Como se pode ver pelo exemplo acima, as regras da aritmética se aplicam sempre, não importando qual seja a fase do seu sistema de numeração. Você pode tentar operações simples num sistema de base 7 ou 8, e assim por diante. Qualquer que seja a base, as regras matemáticas são sempre válidas.

No nosso sistema nonário, a cada casa para a esquerda que o número desloca, seu valor absoluto é multiplicado por nove. Dessa forma, 3275 em nonário é igual a $5 + 7\times9 + 2\times9\times9 + 3\times9\times9\times9$, ou seja, 2417 em decimal. Você deve ter notado que não existe o número 9 no sistema nonário. Ele é representado por 10. Assim, seria correto dizer que, em nonário, 3275 é igual a $5 + 7\times10 + 2\times10\times10 + 3\times10\times10\times10$.

Com um pouco de agilidade mental você pode fazer alguns cálculos simples em nonário. Por exemplo: 99×9 dá 891 em decimal. Em nonário, o número 99 é representado por 120; isto é: $1\times9\times9 + 2\times9$, ou $81+18$. E 9 é 10.

Assim, em nonário, 99×9 se traduz por 120×10 ou 1200, que é $1\times9\times9\times9 + 2\times9\times9$, 891 em decimal.

Qualquer adição, subtração, multiplicação ou divisão de números nonários funcionará normalmente. Confira você mesmo. Só é preciso um pouco de cautela. Lembre-se que 16 em decimal é 17 em nonário.

PROBLEMAS DAS BASES ACIMA DE 10

Lidar com sistemas numéricos de base maior que 10 é um pouco mais difícil do que vimos até agora. Para manejá-los mais facilmente, temos que inventar novos números.

Dessa maneira, se você quisesse empregar um sistema de base 12, teria de inventar números para representar o 10 e o 11. Em duodecimal, 10 representa 1×12, ou 12. E 11 significa $1\times12 + 1$, ou seja, 13 em decimal.

O modo mais fácil de aumentar a faixa dos números é utilizar o alfabeto. O 10 pode ser A, e o 11, B. A2 seria, assim, 122; e BA, 142.

UM PROGRAMA PARA TODAS AS BASES

Fazer contas em bases de numeração que não sejam decimais pode ser bastante difícil e cansativo. Assim, apresentamos aqui um programa que faz contas em qualquer base até 36. Quando você rodá-lo, ele vai perguntar em que base quer trabalhar. Você deve responder em decimal.

A seguir, ele pede um número. Este deve ser digitado na base que você selecionou. Se foi a base 8, não deve haver nenhum número acima de 7. E se você escolheu uma base acima de 10, os números acima de 9 deverão ser digitados por meio das letras do alfabeto, sempre maiúsculas. 10 será A, 11, B, 12, C, e assim por diante.

Então, o computador perguntará que operação você deseja (+, −, * ou / representando respectivamente soma, subtração, multiplicação e divisão). Após isso, você deverá digitar outro número, na mesma base do anterior. Se a sua operação é divisão, e a resposta não é um inteiro, o computador lhe dirá que a operação não será realizada. "Divisão não exata", será a mensagem no vídeo. Se não, a operação e seu resultado aparecerão na tela na base que você escolheu para operar.

```
10 CLS
20 INPUT"BASE(ATE 36)";B
30 INPUT"NUMERO(INTEIRO) ";A$
40 INPUT"SINAL ";S$
50 INPUT"NUMERO(INTEIRO) ";B$
60 P$=A$:GOSUB 210
70 X$=STR$(DE)
80 P$=B$:GOSUB 210
```

```
90 Y$=STR$(DE)
100 IF S$="*" THEN X=VAL(X$)*VA
L(Y$)
110 IF S$="/" THEN X=VAL(X$)/VA
L(Y$)
120 IF S$="+" THEN X=VAL(X$)+VA
L(Y$)
130 IF S$="-" THEN X=VAL(X$)-VA
L(Y$)
140 IF X<>INT(X) THEN N$="IMPOS
SIVEL":GOTO 190
150 U=B*INT(X/B):IF X-U>9 THEN
N$=CHR$(55+(X-U))+N$:GOTO 170
160 N$=MID$(STR$(X-U)+N$,2)
170 X=INT(X/B)
180 IF X>0 GOTO 150
190 PRINT @257,A$+" "+S$+" "+B$
;"= ";N$
200 END
```

MICRO DICAS

UM MODELO PARA NÚMEROS EM DIFERENTES BASES

Às vezes é muito difícil visualizar como funciona um sistema de numeração de base diferente ao de base 10. Afinal, não é fácil ter que cortar fora alguns dedos, ou fazer crescer dedos adicionais nas mãos, de modo que você parece obrigado a se satisfazer com o que tem! Mas seria possível, por exemplo, grudar dois dedos um no outro com fita adesiva, para contar na base 9, ou usar dois palitinhos extras de contagem, para trabalhar na base 12.

Uma solução mais inteligente seria construir o seu próprio ábaco, ou calculadora de continhas, similar àqueles mostrados nas fotos que acompanham esta lição. Um ábaco convencional (mesmo aqueles de brinquedo, para criança) pode ajudar e agilizar as contas no sistema de base 10: assim, você pode criar um ábaco com tantas bolinhas em cada vareta da moldura quantas quiser. Não é necessário nem mesmo construir uma moldura cheia de arames, mas simplesmente arranjar uma série de sulcos paralelos, para conter as bolinhas e fazê-las rolar para cima ou para baixo.

Não se esqueça de que o número de continhas em cada fileira é um a menos do que a base em que você quer trabalhar. Assim, na base 10, você precisa de 9 contas, e na base 2, de apenas uma. Nenhuma bolinha equivale ao número zero, em qualquer sistema de numeração. Uma bolinha afastada das demais significa o número 1, e assim por diante. Quando a operação exigir mais de 9 bolinhas no sistema decimal, zere a fileira de volta, e suba uma bolinha na fileira seguinte (é a operação de "vai-um").

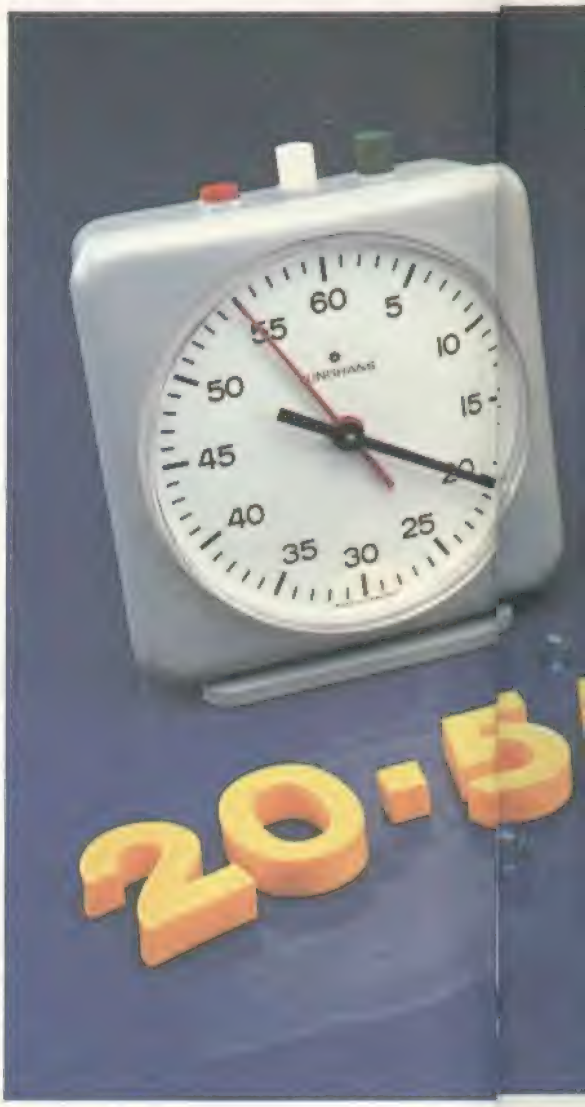
```
210 DE=0
220 IN=0
230 Y=ASC(RIGHT$(P$,1))
240 IF Y<58 THEN D=Y-48:GOTO 26
0
250 D=Y-55
260 DE=DE+D*B^IN
270 P$=LEFT$(P$,LEN(P$)-1)
280 IN=IN+1
290 IF LEN(P$)>0 THEN GOTO 230
300 RETURN
```

Este programa, como está, roda apenas nos compatíveis com o TRS-Color. Para rodá-lo nos modelos TRS-80, modifique a linha 190 para:

```
190 PRINT@513,A$+" "+S$ + "  "
    + B$+ "= ";N$
```

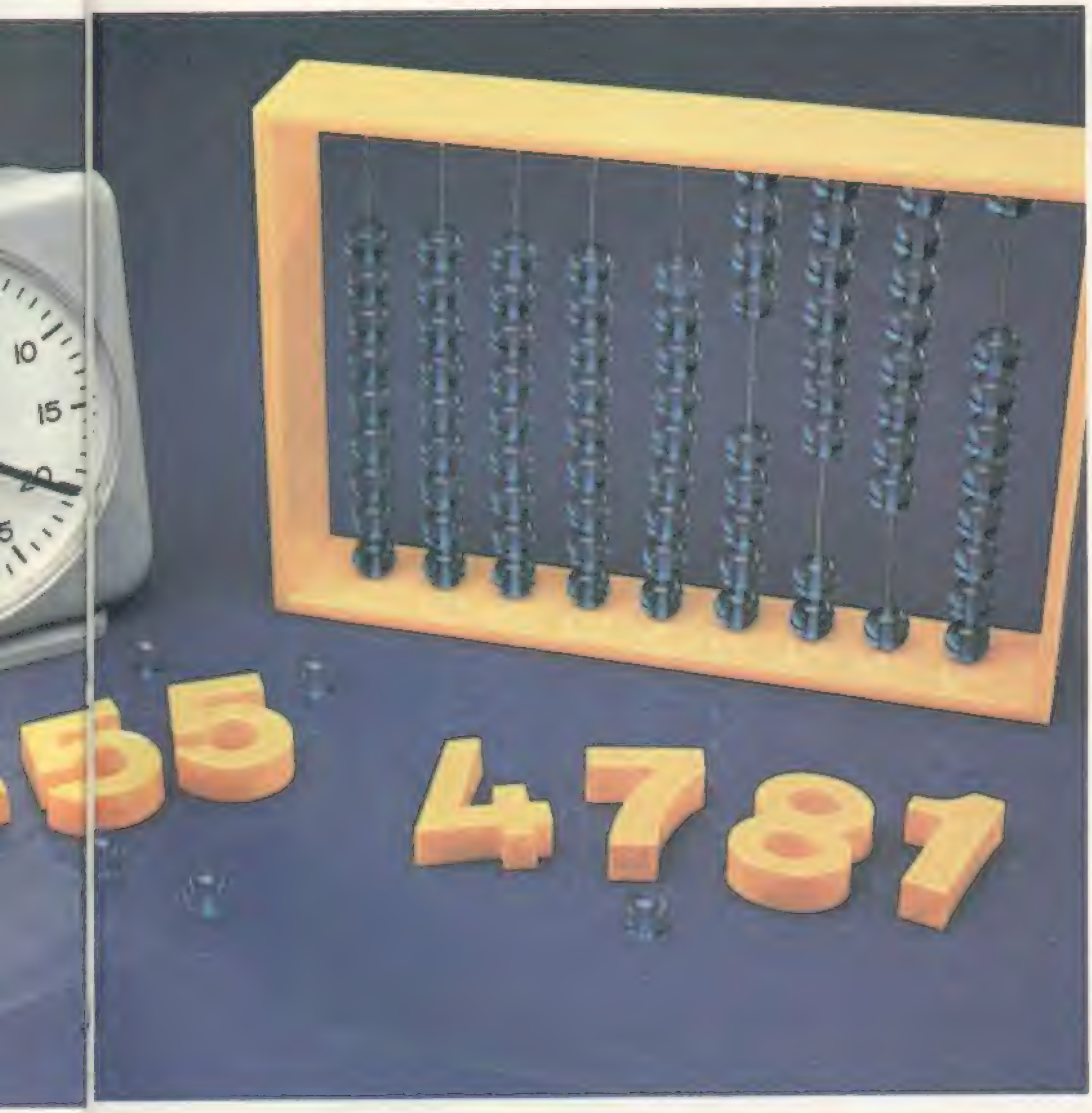
```
 10  POKE 23658,8
 20 INPUT "Base (ate 36) ";b
 30 INPUT "Numero (inteiro) ";
LINE a$
 40 INPUT "Sinal "; LINE s$
 50 INPUT "Numero (inteiro) ";
LINE b$
 60 CLS
 70 LET p$=a$: GOSUB 180
 80 LET x$=STR$ dec
 90 LET p$=b$: GOSUB 180
 95 LET y$=STR$ dec
100 LET z$=x$+s$+y$: LET x=VAL
z$
110 IF x<>INT x THEN  LET n$="
Impossivel realizar a operacao
.": GOTO 160
120 LET n$=""
130 LET u=INT (x/b): LET u=u*b
: IF x-u>9 THEN  LET n$=CHR$ (
55+(x-u))+n$: GOTO 140
135 LET n$=STR$ (x-u)+n$
140 LET x=INT (x/b)
150 IF x>0 THEN  GOTO 130
160 PRINT AT 10,0;a$+s$+b$;"=
";n$
170 STOP
180 LET dec=0
190 LET indice=0
200 LET y=CODE p$(LEN p$)
210 IF y<58 THEN  LET d=y-48:
GOTO 220
215 LET d=y-55
220 LET dec=dec+d*b^indice
230 LET p$=p$(1 TO (LEN p$)-1)
240 LET indice=indice+1
250 IF LEN p$>0 THEN  GOTO 200
260 RETURN
```

```
10 CLS
20 INPUT "BASE (ATÉ 36)";B
30 INPUT "NÚMERO (INTEIRO) ";A$
40 INPUT "SINAL ";S$
50 INPUT "NÚMERO (INTEIRO) ";B$
60 P$=A$:GOSUB 210
70 X$=STR$(DE)
80 P$=B$:GOSUB 210
90 Y$=STR$(DE)
100 IF S$="*" THEN X=VAL(X$)*VA
L(Y$)
110 IF S$="/" THEN X=VAL(X$)/VA
L(Y$)
120 IF S$="+" THEN X=VAL(X$)+VA
L(Y$)
130 IF S$="-" THEN X=VAL(X$)-VA
L(Y$)
140 IF X<>INT(X) THEN N$="Opera
ção impossível":GOTO 190
150 U=B*INT(X/B):IF X-U>9 THEN
N$=CHR$(55+(X-U))+N$:GOTO 170
160 N$=MID$(STR$(X-U)+N$,2)
170 X=INT(X/B)
180 IF X>0 GOTO 150
190 LOCATE 0,12:PRINT A$+" "+S$
+" "+B$+" "+"="+" "+N$
200 END
210 DE=0
220 IN=0
230 Y=ASC(RIGHT$(P$,1))
240 IF Y<58 THEN D=Y-48:GOTO 26
0
250 D=Y-55
260 DE=DE+D*B^IN
270 P$=LEFT$(P$,LEN(P$)-1)
280 IN=IN+1
290 IF LEN(P$)>0 THEN GOTO 230
300 RETURN
```



```
10  HOME
20  INPUT "BASE (ATE 36) =>";B
30  INPUT "NUMERO (INTEIRO) =>"
;A$
40  INPUT "OPERACAO (+,-,*,/) =
>";S$
50  INPUT "NUMERO (INTEIRO) =>"
;B$
60 P$ = A$: GOSUB 210
70 X = DE
80 P$ = B$: GOSUB 210
90 Y = DE
100  IF S$ = "*" THEN X = X * Y
110  IF S$ = "/" THEN X = X / Y
120  IF S$ = "+" THEN X = X + Y
130  IF S$ = "-" THEN X = X - Y
140  IF X <  >  INT (X) THEN N$
 = "NAO DA": GOTO 190
150 U = B *  INT (X / B): IF X
- U > 9 THEN N$ =  CHR$ (55 + (
X - U)) + N$: GOTO 170
160 N$ =  STR$ (X - U) + N$
170 X =  INT (X / B)
180  IF X > 0 THEN 150
190  HTAB 10: VTAB 15: PRINT A$
;" ";S$;" ";B$;" = ";N$
200  END
210 DE = 0
220 IN = 0
230 Y =  ASC ( RIGHT$ (P$,1))
240  IF Y < 58 THEN D = Y - 48:
 GOTO 260
250 D = Y - 55
260 DE = DE + D * B ^ IN
265  IF  LEN (P$) = 1 THEN  RET
URN
270 P$ =  LEFT$ (P$, LEN (P$) -
 1)
280 IN = IN + 1
290  GOTO 230
300  RETURN
```

Se você acha que fazer contas em diversas bases é mais do que pretende saber algum dia, note que o computador também trapaceia. O programa, na realidade, converte os números que você digitou em decimais, executa a operação e reconverte o resultado para a base que você escolheu. Ao contrário da mente humana, o computador só consegue fazer cálculos em decimal, dentro de um programa escrito em BASIC, apesar de converter os números para binário, quando as instruções são traduzidas para código de máquina durante a execução do programa.



E AGORA, BINÁRIOS

Para os computadores digitais, o sistema de numeração mais apropriado é o baseado no número 2. Isso acontece porque o computador é composto de circuitos eletrônicos que possuem dois estados evidentes: ligado e desligado. O estado "desligado" representa o número 0, e o "ligado", o número 1. E, na base 2, estes são todos os números de que você necessita para expressar qualquer valor.

O sistema de numeração de base 2 é conhecido como "binário". Ele é composto apenas por zeros e uns; todos os outros números foram abolidos. Dessa forma, se você começar a contar da maneira habitual, não vai passar de 1. Some 1 a 1 e terá o número 1 deslocado uma casa para a esquerda, enquanto a casa da direita é zerada. Assim, em binário, 1 + 1 = 10.

Contar de 0 a 8 resulta em 0, 1, 10, 11, 100, 101, 110, 111, 1000. Novamente, estes números obedecem às leis da aritmética. Se você quiser somar 10+11, deve primeiramente somar os dois números da direita, 0+1. A seguir, devem ser somados os dois da esquerda, 1+1, o que dá 2; só que 2 em binário é 10. Assim 10+11 = 101; ou seja, 2+3 = 5.

Subtração em binário é um processo tão direto quanto a soma. O único detalhe a observar é o número que se "leva" para a próxima casa.

A multiplicação e a divisão são extraordinariamente fáceis, mesmo para seres humanos "normais". Na realidade, é tão fácil que "até" um computador pode fazer essas operações. No primeiro caso, tudo se resume a multiplicar 0×0, que dá 0; 0×1, que também dá 0; ou 1×1, que dá 1.

A divisão é igualmente simples: em cada operação, basta decidir se o divisor está dentro do dividendo uma vez ou nenhuma vez.

Confira você mesmo como os processos aritméticos funcionam em binário. O painel das páginas seguintes dá exemplos de operações para você fazer.

Qualquer sistema de numeração pode ser imitado por um modelo mecânico: o relógio, por exemplo, usa a base 60, enquanto o ábaco utiliza a base 10.

BITS E BYTES

Os números binários refletem exatamente o que acontece no computador. As instruções ou dados que você dá a ele são codificados em números binários, que, por sua vez, são manipulados e armazenados pelos circuitos internos do aparelho. Assim, quando você começa a dominar o sistema binário, começa também a compreender como funciona o computador.

Cada um dos dígitos de um número binário é representado eletronicamente dentro do computador por um circuito ligado ou desligado. Se está ligado, o valor é 1. Se, pelo contrário, o circuito está desligado, o valor é 0. Cada dígito é conhecido como um *bit*.

Esses circuitos são agrupados em unidades maiores, que representam números também maiores e mais úteis. Em quase todos os micros, os bits estão organizados em grupos de oito, formando o que se conhece como um *byte*. Cada byte representa, assim, oito dígitos binários, de forma que pode armazenar qualquer número entre 00000000, ou zero, e 11111111, ou 128 + 64 + 32 + 16 + 8 + 4 + 2 + 1, que é 255. Números maiores que 255 são representados por dois ou mais bytes.

FRAÇÕES BINÁRIAS

Todos os programas e exemplos dados até aqui manipularam números inteiros. É possível, também, converter frações em números binários.

Por exemplo, 1/2 é 0.1 em binário; 1/4 é 0.01 e 1/8 é 0.001. Existe um padrão determinado. Qualquer fração pode ser formada a partir de 1/2s, 1/4s, 1/8s, 1/16s, etc., da mesma forma que qualquer número inteiro pode ser formado a partir de 1s, 2s, 4s, 8s, etc. O problema é que é mais difícil trabalhar com frações binárias porque sempre temos uma infinidade de 0s e 1s!

Quando você converte frações, é melhor pensar nelas como 0.5; 0,25 e assim por diante. Desta forma, você pode manipular uma fração como 0.75 como sendo $1 \times 0.5 + 1 \times 0.25$, o que significa em binário 0.11.

O próximo programa permite que você trabalhe com frações binárias sem grandes dificuldades:

```
10 CLS
20 PRINT"CONVERSAO DE DECIMAL P
/ BINARIO"
30 PRINT:INPUT"DIGITE UM NUMERO
 ENTRE 0 E 1    ";N
40 IF N<=0 OR N>=1 THEN 30
50 N$="0."
60 FOR T=1 TO 30
70 N=N*2
80 N$=N$+CHR$(48+INT(N))
90 N=N-INT(N)
100 NEXT T
110 PRINT @257,"O NUMERO BINARI
O E: "
120 PRINT:PRINT N$:PRINT
130 PRINT"OUTRO NUMERO (S/N) ?"
140 A$=INKEY$
150 IF A$="S" THEN 10
160 IF A$<>"N" THEN 140
170 END
```

O programa acima, como está, roda apenas nos compatíveis com o TRS-Color. Para os modelos tipo TRS-80, modifique a linha 110 para:

```
110 PRINT@513,"O NUMERO BINARIO
    E' :-"
```

```
 10 CLS
 20 PRINT '"Conversao de Decim
 al para Binario"
 30 INPUT "Digite um numero en
 tre 0 e 1",N
 40 IF N<=0 OR N>=1 THEN  GOTO
 30
 45 PRINT ''"Decimal: ";N
 50 LET N$="0."
 60 FOR T=1 TO 16
 70 LET N=N*2+1E-9
 80 LET N$=N$+CHR$ (48+INT N)
 90 LET N=N-INT N
100 NEXT T
110 PRINT ''"O numero binario
e:"
120 PRINT N$
130 PRINT '''''"Outro numero (
S/N)?"
140 LET A$=INKEY$
150 IF A$="S" OR A$="s" THEN
GOTO 10
160 IF A$<>"N" AND A$<>"n"
THEN  GOTO 140
```

Tal como está, o programa acima roda apenas no Sinclair Spectrum

```
ADIÇAO BINARIA          DECIMAL

 1100111                1100111 = 103

 +100010                 100010 =  34
 -------
10001001               103 + 34 = 137

                       10001001 = 137
```

A adição binária funciona a partir da direita. 0 + 0 dá 0, 0 + 1 dá 1, e 1 + 1 resulta em 10, que é o 0 com "vai-um" para o próximo dígito. Observe como essas operações funcionam na soma realizada acima.

```
SUBTRAÇAO BINARIA        DECIMAL

 1011001                1011001 =  89

 -110011                 110011 =  51
 --------
  100110                89 - 51 =  38

                         100110 =  38
```

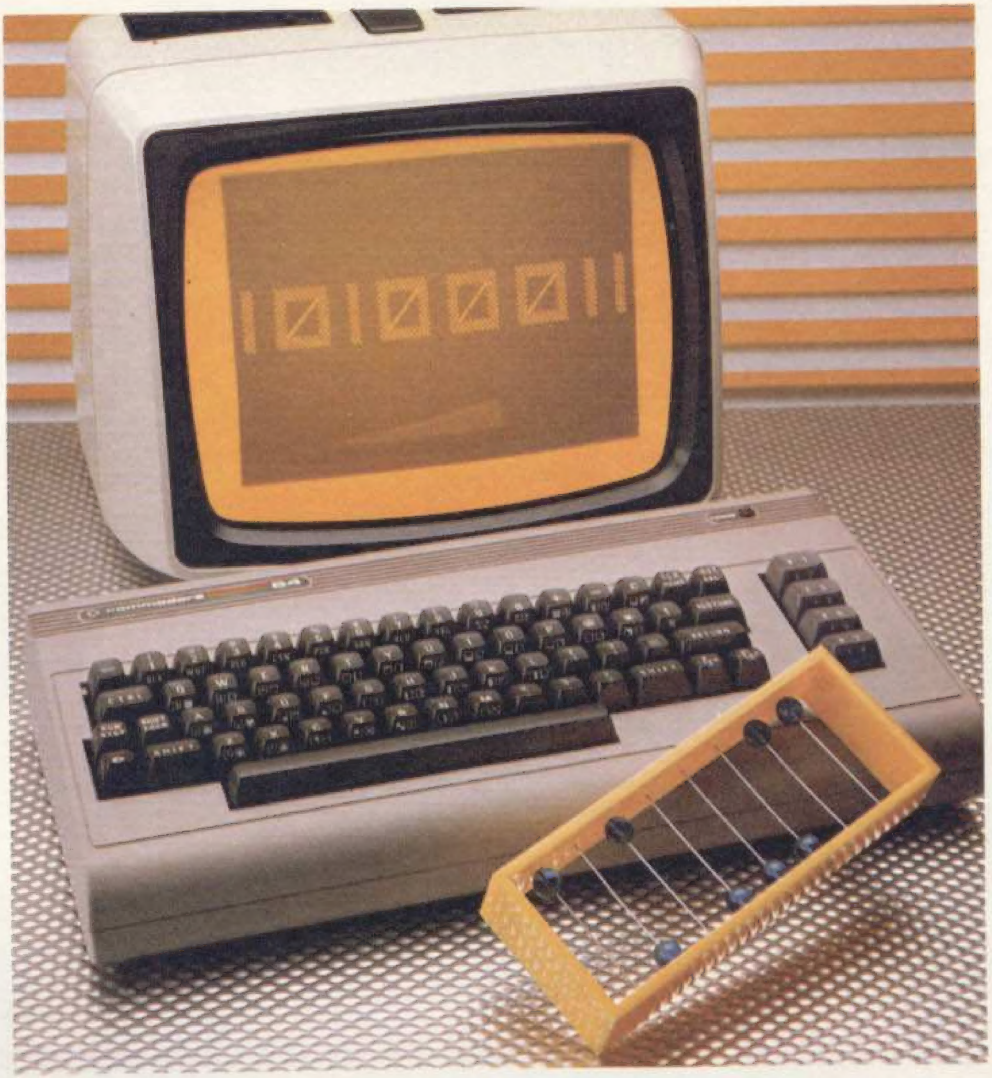
A subtração binária é igualmente simples. 1 − 1 dá 0, 0 − 1 dá 1. Agora, a operação 1 − 0 dá 0 e "vai-um" para o próximo dígito à esquerda. Os resultados, portanto, não são todos exatamente iguais ao da mesma operação no sistema decimal.

(TK-90X). Para rodar no ZX-81, substitua as linhas seguintes:

```
30 PRINT "ENTRE UM NUMERO ENTRE
   0 E 1"
35 INPUT N
80 LET N$=N$+CHR$(28+INT N)
```

```
10 CLS
20 PRINT TAB(7);"BINÁRIO >> DEC
IMAL"
30 PRINT:INPUT " digite um núme
ro entre 0 e 1 ";N
40 IF N<=0 OR N>=1 THEN 30
50 N$="0."
60 FOR T=1 TO 30
70 N=N*2
80 N$=N$+CHR$(48+INT(N))
90 N=N-INT(N)
100 NEXT T
110 LOCATE 0,10:PRINT "O número
 binário é : "
120 PRINT:PRINT N$:PRINT
130 PRINT "outro número (S/N) ?
"
140 A$=INKEY$
150 IF A$="S" THEN 10
160 IF A$<>"N" THEN 140
170 END
```

```
10  HOME
20  HTAB 5: PRINT "CONVERSAO DE
```



Um ábaco binário utiliza uma conta em cada vareta. No computador, essa disposição é substituída por pulsos elétricos.

```
MULTIPLICACAO BINARIA    DECIMAL

        1101                1101 =  13
x       1011                1011 =  11
------------
        1101             13 x 11 = 143
       1101             10001111 = 143
      0000
      1101
------------
    10001111
```

Em cada fase da multiplicação binária, você multiplica 0 por 0, que dá 0, ou 1 por 0, que também dá 0, ou ainda 1 por 1, que resulta sempre em 1. Veja como isto se aplica no exemplo acima, onde é simulada uma operação no vídeo do computador.

```
DIVISAO BINARIA                DECIMAL

   11001 | 101                 11001 =  25
         |---
   101     101                   101 =   5
    000                       25 / 5 =   5
     101                          101 =   5
```

Na divisão binária, finalmente, é necessário verificar se o divisor cabe no dividendo pelo menos uma vez. Se isto acontecer, o resultado será 1; caso contrário, será 0. Isso faz da divisão uma das operações binárias mais simples.

```
CIMAL PARA BINARIO"
30  PRINT : PRINT "DIGITE UM NU
MERO": INPUT "ENTRE 0 E 1 =>";N
40  IF N <  = 0 OR N >  = 1 THE
N 30
50 N$ = "0."
60  FOR T = 1 TO 38
70 N = N * 2
80 N$ = N$ +  CHR$ (48 +  INT (
N))
90 N = N -  INT (N)
100  NEXT T
110  HTAB 1: VTAB 15: PRINT "O
NUMERO BINARIO E: "
120  PRINT : PRINT N$: PRINT
130  PRINT : PRINT "OUTRO NUMER
O? ";
140  GET A$
150  IF A$ = "S" THEN 10
160  IF A$ <  > "N" THEN 130
170  END
```

O programa aceita qualquer número entre 0 e 1 e imprime o seu equivalente binário. A linha 70 começa por dobrar o seu número e a linha 80 constrói o número binário. Inicialmente o computador trabalha com o valor de INT(N), que será ou 0 ou 1. Então, a esse valor adiciona-se 48 para perfazer 48 ou 49, que são os códigos ASCII para 0 e 1. Tudo isto é feito de forma a transformar os números em cadeias de caracteres para que possam ser adicionados à variável **N$**. Na linha 90 tiramos a parte inteira de N para deixar apenas a decimal.

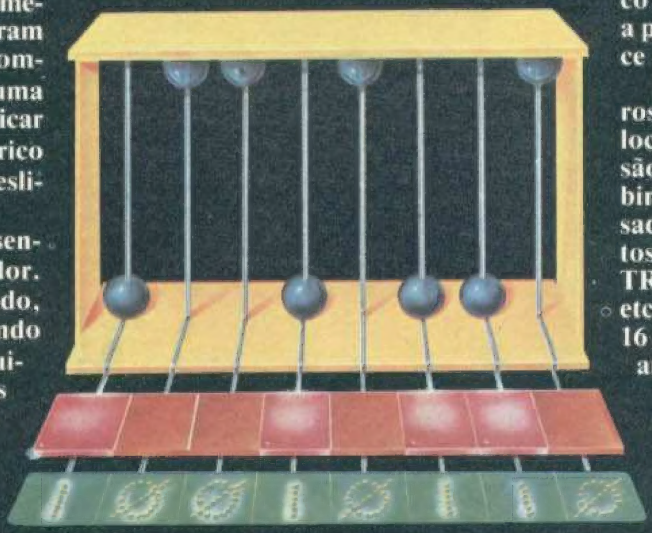
O programa no Sinclair Spectrum é um pouco diferente, porque enquanto INT 1 é 1, como se espera, INT (.5×2) é 0. Isto acontece porque ele armazena .5×2 como .99999999... Assim, uma pequena fração tem que ser adicionada a N na linha 70 para fazer com que INT funcione como deve.

Nos modelos compatíveis com o ZX-81, a linha 80 usa outro número para somar ao valor inteiro de N (observe que essa máquina não usa o código ASCII). O valor 28 corresponde ao caractere 0, o valor 29 ao caractere 1, etc. e assim sucessivamente.

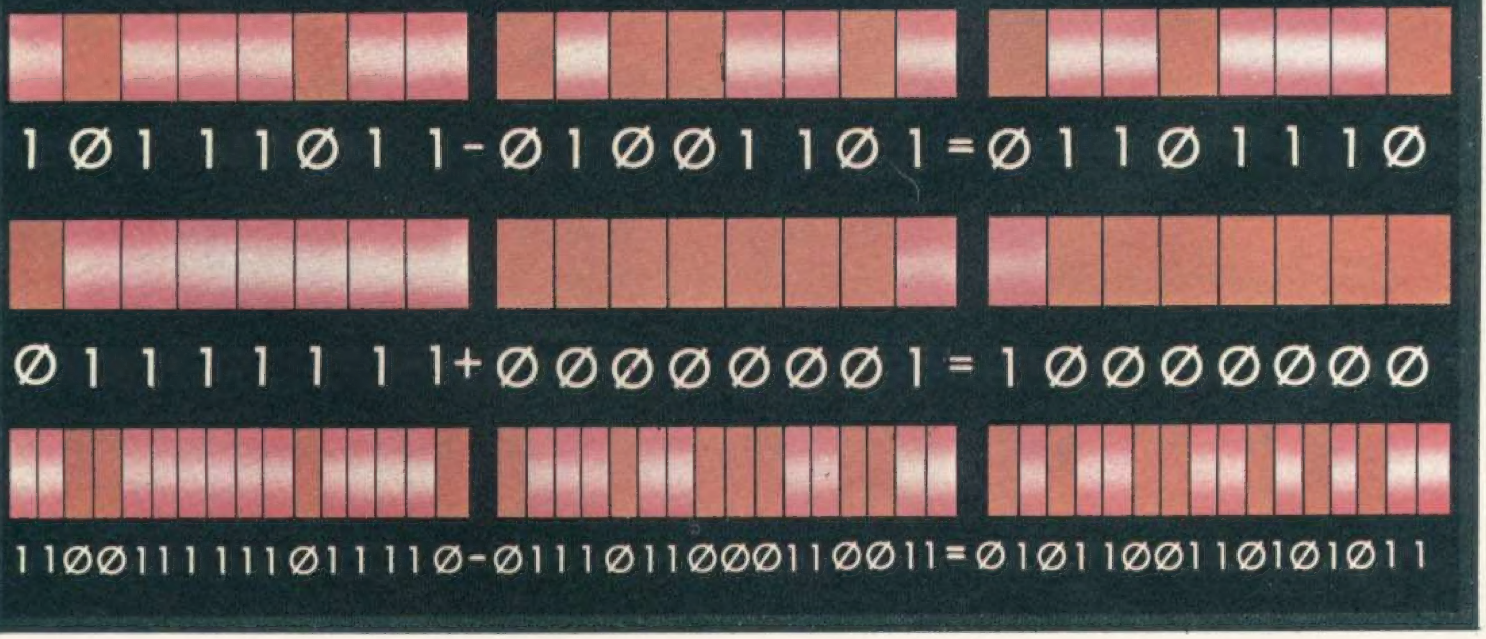
O LONGO TRAJETO DO ÁBACO À ELETRÔNICA

Os números binários podem ser adicionados, subtraídos, multiplicados e divididos como qualquer número na base 10. Entretanto, eles foram escolhidos para utilização nos computadores digitais em virtude de uma propriedade singular: é fácil verificar se um determinado circuito elétrico ou eletrônico está ligado ou desligado.

Os dígitos binários são representados por circuitos de computador. Quando um circuito está desligado, ele representa o dígito 0, e quando está ligado, o dígito 1. Nos circuitos integrados do computador, os circuitos binários são arranjados em grupos de 8, de tal forma que valores binários de 8 bits podem ser representados. Isso dá uma gama de representação de 00000000 — ou 0 em decimal — a 11111111 — ou 255 em decimal. No manual do seu computador esse "número mágico" — 255 (ou 256, se você contar a partir de 1, ao invés de 0) — aparece a todo momento.



Às vezes são necessários números binários muito maiores, e duas locações de memória de 8 bits cada são juntadas para obter um número binário de 16 bits. O microprocessador Zilog Z-80, existente em muitos computadores pessoais, como o TRS-80, os compatíveis com MSX, etc., pode trabalhar com números de 16 bits, e até mesmo fazer operações aritméticas com eles. Nestes casos, números binários entre 0000000000000000 e 1111111111111111, ou seja, entre 0 e 65.535, em decimal, podem ser manipulados.



CLASSIFICAÇÃO DOS MICROCOMPUTADORES POR LINHA

LINHA	FABRICANTE	MODELO
Apple II +	Appletronica	Thor 2010
Apple II +	CCE	MC-4000 Exato
Apple II +	CPA	Absolutus
Apple II +	CPA	Polaris
Apple II +	Digitus	DGT-AP
Apple II +	Dismac	D-8100
Apple II +	ENIAC	ENIAC II
Apple II +	Franklin	Franklin
Apple II +	Houston	Houston AP
Apple II +	Magnex	DM II
Apple II +	Maxitronica	MX-2001
Apple II +	Maxitronica	MX-48
Apple II +	Maxitronica	MX-64
Apple II +	Maxitronica	Maxitronic I
Apple II +	Microcraft	Craf II Plus
Apple II +	Milmar	Apple II Plus
Apple II +	Milmar	Apple Master
Apple II +	Milmar	Apple Senior
Apple II +	Omega	MC-400
Apple II +	Polymax	Maxxi
Apple II +	Polymax	Poly Plus
Apple II +	Spectrum	Microengenho I
Apple II +	Spectrum	Spectrum ed
Apple II +	Suporte	Venus II
Apple II +	Sycomig	SIC I
Apple II +	Unitron	AP II
Apple II +	Victor do Brasil	Elppa II Plus
Apple II +	Victor do Brasil	Elppa Jr.
Apple IIe	Microcraft	Craft IIe
Apple IIe	Microdigital	TK-3000 IIe
Apple IIe	Spectrum	Microengenho II
MSX	Gradiente	Expert GPC-1
MSX	Sharp	Hotbit HB-8000
Sinclair Spectrum	Microdigital	TK-90X
Sinclair Spectrum	Timex	Timex 2000
Sinclair ZX-81	Apply	Apply 300
Sinclair ZX-81	Engebras	AS-1000
Sinclair ZX-81	Filcres	NEZ-8000
Sinclair ZX-81	Microdigital	TK-82C
Sinclair ZX-81	Microdigital	TK-83
Sinclair ZX-81	Microdigital	TK-85
Sinclair ZX-81	Prologica	CP-200
Sinclair ZX-81	Ritas	Ringo R-470
Sinclair ZX-81	Timex	Timex 1000
Sinclair ZX-81	Timex	Timex 1500
TRS-80 Mod. I	Dismac	D-8000
TRS-80 Mod. I	Dismac	D-8001/2
TRS-80 Mod. I	LNW	LNW-80
TRS-80 Mod. I	Video Genie	Video Genie I
TRS-80 Mod.III	Digitus	DGT-100
TRS-80 Mod.III	Digitus	DGT-1000
TRS-80 Mod.III	Kemitron	Naja 800
TRS-80 Mod.III	Prologica	CP-300
TRS-80 Mod.III	Prologica	CP-500
TRS-80 Mod.III	Sysdata	Sysdata III
TRS-80 Mod.III	Sysdata	Sysdata Jr.
TRS-80 Mod.IV	Multix	MX-Compacto
TRS-80 Mod.IV	Sysdata	Sysdata IV
TRS-Color	Codimex	CS-6508
TRS-Color	Dynacom	MX-1600
TRS-Color	LZ	Color 64
TRS-Color	Microdigital	TKS-800
TRS-Color	Prologica	CP-400

CLASSIFICAÇÃO DOS MICROCOMPUTADORES POR NOME

FABRICANTE	MODELO	PAÍS	LINHA
Appletronica	Thor 2010	Brasil	Apple II +
Apply	Apply 300	Brasil	Sinclair ZX-81
CCE	MC-4000 Exato	Brasil	Apple II +
CPA	Absolutus	Brasil	Apple II +
CPA	Polaris	Brasil	Apple II +
Codimex	CS-6508	Brasil	TRS-Color
Digitus	DGT-100	Brasil	TRS-80 Mod.III
Digitus	DGT-1000	Brasil	TRS-80 Mod.III
Digitus	DGT-AP	Brasil	Apple II +
Dismac	D-8000	Brasil	TRS-80 Mod. I
Dismac	D-8001/2	Brasil	TRS-80 Mod. I
Dismac	D-8100	Brasil	Apple II +
Dynacom	MX-1600	Brasil	TRS-Color
ENIAC	ENIAC II	Brasil	Apple II +
Engebras	AS-1000	Brasil	Sinclair ZX-81
Filcres	NEZ-8000	Brasil	Sinclair ZX-81
Franklin	Franklin	USA	Apple II +
Gradiente	Expert GPC1	Brasil	MSX
Houston	Houston AP	Brasil	Apple II +
Kemitron	Naja 800	Brasil	TRS-80 Mod.III
LNW	LNW-80	USA	TRS-80 Mod. I
LZ	Color 64	Brasil	TRS-Color
Magnex	DM II	Brasil	Apple II +
Maxitronica	MX-2001	Brasil	Apple II +
Maxitronica	MX-48	Brasil	Apple II +
Maxitronica	MX-64	Brasil	Apple II +
Maxitronica	Maxitronic I	Brasil	Apple II +
Microcraft	Craft II Plus	Brasil	Apple II +
Microcraft	Caft IIe	Brasil	Apple IIe
Microdigital	TK-3000 IIe	Brasil	Apple IIe
Microdigital	TK-82C	Brasil	Sinclair ZX-81
Microdigital	TK-83	Brasil	Sinclair ZX-81
Microdigital	TK-85	Brasil	Sinclair ZX-81
Microdigital	TK-90X	Brasil	Sinclair Spectrum
Microdigital	TKS-800	Brasil	TRS-Color
Milmar	Apple II Plus	Brasil	Apple II +
Milmar	Apple Master	Brasil	Apple II +
Milmar	Apple Senior	Brasil	Apple II +
Multix	MX-Compacto	Brasil	TRS-80 Mod.IV
Omega	MC-400	Brasil	Apple II +
Polymax	Maxxi	Brasil	Apple II +
Polymax	Poly Plus	Brasil	Apple II +
Prologica	CP-200	Brasil	Sinclair ZX-81
Prologica	CP-300	Brasil	TRS-80 Mod.III
Prologica	CP-400	Brasil	TRS-Color
Prologica	CP-500	Brasil	TRS-80 Mod.III
Ritas	Ringo R-470	Brasil	Sinclair ZX-81
Sharp	Hotbit HB-8000	Brasil	MSX
Spectrum	Microengenho I	Brasil	Apple II +
Spectrum	Microengenho II	Brasil	Apple IIe
Spectrum	Spectrum ed	Brasil	Apple II +
Suporte	Venus II	Brasil	Apple II +
Sycomig	SIC I	Brasil	Apple II +
Sysdata	Sysdata III	Brasil	TRS-80 Mod.III
Sysdata	Sysdata IV	Brasil	TRS-80 Mod.IV
Sysdata	Sysdata Jr.	Brasil	TRS-80 Mod.III
Timex	Timex 1000	USA	Sinclair ZX-81
Timex	Timex 1500	USA	Sinclair ZX-81
Timex	Timex 2000	USA	Sinclair Spectrum
Unitron	AP II	Brasil	Apple II +
Victor do Brasil	Elppa II Plus	Brasil	Apple II +
Victor do Brasil	Elppa Jr.	Brasil	Apple II +
Video Genie	Video Genie I	USA	TRS-80 Mod. I

UM LOGOTIPO PARA CADA MODELO DE COMPUTADOR

INPUT foi especialmente projetado para microcomputadores compatíveis com as sete principais linhas existentes no mercado.
Os blocos de textos e listagens de programas aplicados apenas a determinadas linhas de micros podem ser identificados por meio dos seguintes símbolos:

Sinclair ZX-81

TRS-80

TK-2000

MSX

Spectrum

TRS-Color

Apple II

Quando o emblema for seguido de uma faixa, então tanto o texto como os programas que se seguem passam a ser específicos para a linha indicada.

NO PRÓXIMO NÚMERO

PROGRAMAÇÃO BASIC

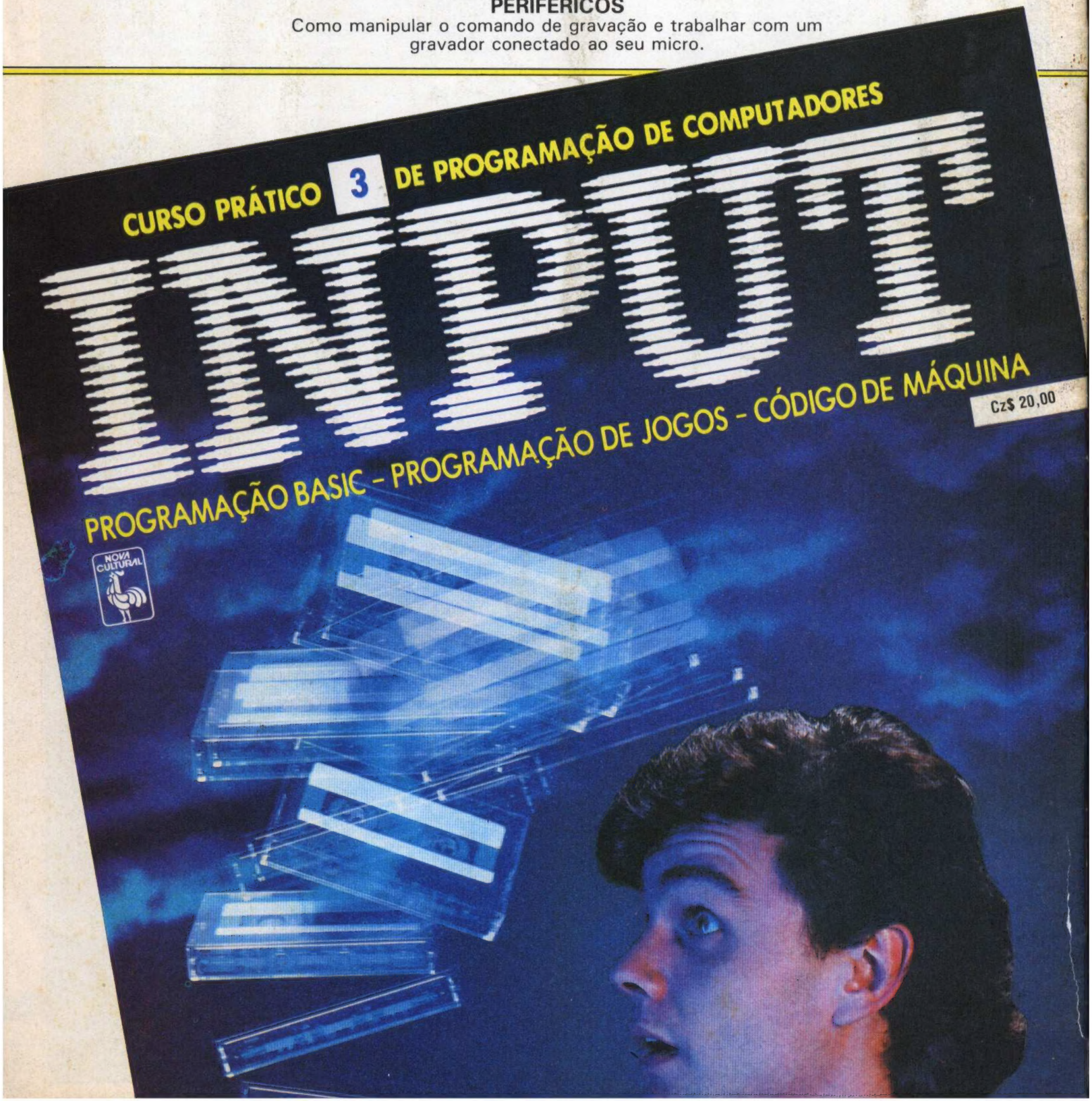
O computador funciona de forma lógica, embora não tenha cérebro. Para isso, basta que você o programe de modo adequado.

CÓDIGO DE MÁQUINA

Aprenda aritmética hexadecimal e domine o código de máquina.

PROGRAMAÇÃO DE JOGOS

Programe jogos, movendo figuras em labirintos. Construa paredes intransponíveis e encontre o mapa do tesouro.

PERIFÉRICOS

Como manipular o comando de gravação e trabalhar com um gravador conectado ao seu micro.